

A SUPPORT SYSTEM FOR OPTIMIZATION MODELLING

**A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of**

DOCTOR OF PHILOSOPHY

by

INDU SHEKHAR SINGH

to the

**INDUSTRIAL AND MANAGEMENT ENGINEERING PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY KANPUR**

JANUARY, 1987

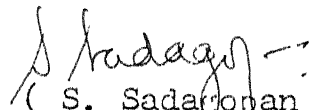
106302

IMEP-1987-D SIN-SUP

5187

CERTIFICATE

This is to certify that the work embodied in the thesis, "A SUPPORT SYSTEM FOR OPTIMIZATION MODELLING", by Mr. Indu Shekhar Singh has been carried out under my supervision and has not been submitted elsewhere for a degree.


(S. Sadagopan)
Assistant Professor
Industrial and Management Engg. Program
Indian Institute of Technology,
Kanpur 208 016

January, 1987

ACKNOWLEDGEMENTS

First acknowledgement is the silent communication to The Supreme!

Next, I am deeply grateful to my thesis supervisor, ^aSadgopanji for his intimate guidance and counsel in all walks of my life during the entire span of my thesis work and stay at IIT Kanpur.

Further, I acknowledge my sincere thanks to Quality Improvement Program of the Dept. of Education, Ministry of Human Resources Development, Govt. of India and the sponsoring Institution, Regional Institute of Technology, Jamshedpur for their kind assistance towards the completion of this research. I am also thankful to the host Institution, IIT Kanpur for extending to me the excellent facilities of the Computer Center, Central Library and Hall of Residence V.

In particular, I express my deep sense of gratitude to Drs. J.L. Batra, A.K. Mittal, Kripa Shanker, A.P. Sinha and R.K. Ahuja for their timely help and encouragement; Dr. G.Barua for rendering all the possible help in the use of the UPTRON machine.

I wish to express my gratitude to the following individuals: Mr. A. Muralidharan for proof reading assistance; Swami Anand Chaitanya for excellent typing; Mr. B.R.Kandiyal for immaculate mimeographic services; M/S P.D. Gupta, G.L.Arora

B.Singh and Amrit Lal for their help in various ways;
Mr. J. Senthil for useful company; and Mr. M. Oja for helping
me in a variety of ways.

I am also thankful to Mr. N. Ravi and all other
friends of CMC and Computer Center for their assistance.
I am deeply thankful to Sri alongwith other family members
for their help and care.

I am heartily thankful to the entire campus community
in general and Satya Sai Devotee family in particular for
providing the soothing environment for me during the entire
span of my stay at the campus.

Lastly, I express my deep gratitude to all members
of my family (including my late uncle) for their sacrifices
during my absence.

Once again, I express my gratitude in silence to
The Supreme!

I. Shukhar

January, 1987.

CONTENTS

<u>Chapter</u>		<u>Page</u>
	CERTIFICATE	
	ACKNOWLEDGEMENTS	
	LIST OF TABLES	
	LIST OF FIGURES	
	LIST OF APPENDICES	
	SYNOPSIS	
I.	INTRODUCTION	1
	1.1 Decision Support Systems	1
	1.2 Optimization Modelling	4
	1.3 Outline of the Thesis	8
II.	LITERATURE SURVEY	10
	2.1 Introduction	10
	2.2 Problem Processor	13
	2.2.1 Basic Features	15
	2.2.2 Current Trends	18
	2.3 Matrix Generators	19
	2.4 Modelling Languages	24
	2.5 Data Management	31
	2.5.1 Introduction	31
	2.5.2 Data Model	32
	2.5.3 Relational Data Model	33
	2.5.4 Query Languages	34
	2.6 Conclusions	37

<u>Chapter</u>		<u>Page</u>
III.	MODELLING SYSTEM LAMP	38
	3.1 Introduction	38
	3.2 Research Goals	40
	3.3 Research Benefits	43
IV.	MODELLING LANGUAGE FEATURES OF LAMP	46
	4.1 Introduction	46
	4.2 Design Goals	53
	4.3 Implementation	60
	4.3.1 Introduction	60
	4.3.2 Grammar	61
	4.3.3 Salient Features	66
	4.4 Illustrative Example	71
	4.4.1 Sample Problem	71
	4.4.2 LAMP Formulation of the Sample Problem	74
	4.5 Further Examples	81
V.	STANDARD INTERFACE FOR DATA MANAGEMENT	86
	5.1 Introduction	86
	5.2 Database Management Systems	87
	5.3 Relational Database Management Systems	90
	5.4 Interface Design	97
VI.	DATA MANAGEMENT ASPECTS OF LAMP	108
	6.1 Introduction	108
	6.2 Data Analysis	111
	6.2.1 Introduction	111
	6.2.2 Input Data Analysis	112
	6.2.3 Output Analysis	124
	6.2.4 "What If" Analysis	129

<u>Chapter</u>		<u>Page</u>
VII.	SUMMARY, CONCLUSIONS AND RECOMMENDATIONS	139
	7.1 Summary of Work	139
	7.2 Contributions	140
	7.3 Limitations	141
	7.4 Recommendations for Further Work	141
	REFERENCES	
	APPENDICES	142

LIST OF TABLES

<u>Table</u>	<u>Page</u>
1. Land Availability	72
2A. Average Unit Labour Requirement	72
2B. Labour Availability	72
3A. Unit Water Requirement	73
3B. Water Available for Crops	73
3C. Total Water Available	73
4A. Unit Fertilizer Requirement	73
4B. Fertilizer Availability	73
5. Unit Profit Contribution	73
6A. Bounds for Crops	74
6B. Bounds for Livestock	74
7A. Yield Data	74
7B. Minimum Production	74

LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
1.	EBNF Grammar for LAMP	64
2.	Agricultural Problem Terminology Phase	75
3.	Agricultural Problem Model Structure	76
4.	Model Formulation in MPS Format for Agricultural Problem	82
5.	Model Formulation in Expanded Form for Agricultural Problem	84
6.	Schema Listing for Input Relations	102
7.	Schema Listing for Output Relations.	103
8.	Sample Input Relations for Agricultural Model	104
9.	Sample Output Relations for Agricultural Model	107
10.	Queries for Problem Statistics	114
11.	Queries for Problem Summary	117
12.	Queries for Simple Error Checking	119
13.	Queries for Sophisticated Checking	122
14.	Queries for Solution Analysis	126
15.	Queries for Sophisticated Analysis	128
16.	Queries for "What If " Analysis	130

LIST OF APPENDICES

Appendix

- A FURTHER EXAMPLES
- B LAMP HELP FILE
- C LAMP DOCUMENTATION
- D LINEAR INTERACTIVE AND DISCRETE OPTIMIZER (LINDO)
- E UNIFY RELATIONAL DATABASE MANAGEMENT SYSTEM
- F STRUCTURED QUERY LANGUAGE (SQL)

SYNOPSIS

The emerging area of Decision Support Systems(DSS) is significantly influenced by the ready availability of micro/personal computers and the recent advances in database management. The widespread use of spreadsheet analysis has involved the managers directly in the decision making process. It is expected that the next generation of DSS would make use of the sophisticated techniques of Operations Research/Management Science (OR/MS). Such a scenario would integrate the user, the display, the interface software, data management and model management systems. This would facilitate the best use of the analytical capabilities of the decision maker and the accurate and fast data retrieval capabilities of the computer. Ultimately the modelling activity itself need to be supported with an integrated system. This dissertation is an attempt in this direction, in the area of optimization modelling.

Optimization modelling using mathematical programs has been receiving an increasing attention in the recent years. This is due to the fact that many real life problems can be easily modelled; they also enjoy a good algorithmic support and efficient implementations. However, the support from the machine to the modeller has been restricted primarily to the numerical solution. The other aspects of modelling such

as, formulation, data analysis and solution analysis have received only a limited attention. Mostly this has been in the form of matrix generators and report writers. Many of them are implementation specific and lack generality.

In this thesis we develop an integrated modelling and data management system, LAMP, that attempts to provide the modeller, support in the entire life cycle of modelling process. Our approach has been to conceptualize an architecture that encompasses all the activities of the modelling phase in situations using mathematical programming. LAMP is designed with a general algebraic notation natural to mathematical programs. Features like indexing, symbolic representation of entities (constraints, variables and objective), symbolic summation & special structures have been implemented. The LAMP has advantages of verifiability, modifiability, documentability and simplicity. The major advantage of such a modelling language has been the independence of problem structure and problem data, thereby enabling the modeller, the meta-level interface with the problem solving software.

LAMP attempts to integrate the modelling language with the other phases of modelling through the concept of relational database management. The input data and the results are viewed as a set of normalized relations. This approach facilitates an elegant structure that can be used as a standard data representation. Incidentally this can serve

as a neat alternative to the industry standard MPS Format. The power of relational database management and standard query languages can be exploited to assist the data analysis, data validation, solution analysis and report writing phases of the modelling with an ease and elegance that far surpass the power of matrix generators and report writers. The data independence and integrity features permit the structure to be implementation independent. Hopefully, such an architecture would pave the way for a new generation of problem solving systems, in both the area of mathematical programming as well as in other areas of management science. The ideas are illustrated using an agricultural planning model.

The implementation is based on the problem processing software LINDO and UNIFY relational data management software using SQL query language.

CHAPTER I

INTRODUCTION

1.1 DECISION SUPPORT SYSTEMS:

The application of computers to managerial problem solving has witnessed a considerable growth in the last three decades, both in quantitative and qualitative terms. The quantitative expansion is readily appreciated through the massive investments being made by corporations, in hardware as well as software, in terms of office automation, computerized planning and control systems and communication equipment. The qualitative improvement though subtle, is in fact the more important one. The earlier days were marked by the dominance of data processing equipment which primarily emphasized mechanization of record keeping and accounting functions. The second generation saw a shift in emphasis from electronic data processing to management information system (MIS). The philosophy of management information system laid stress on the use of information for decision making. In addition it also saw the emergence of models, primarily in the areas of Operations Research/Management Science (OR/MS) to help the managerial decision making activity. Optimization techniques with emphasis on

efficiency were increasingly applied leading to optimal decision rules. The early success of this approach did contribute substantially to the emergence and growth of management science.

Despite the tremendous growth in computer-related activities, the field of Management Information System, had little significant impact on the style and/or process of managerial decision making. This can be traced in large part to the lack of proper perspective on the problems involved in augmenting the decision making ability of management [118]. The philosophy of decision support is primarily intended to overcome this limitation.

This new approach, generally known as Decision Support System (DSS) philosophy, shifts the emphasis from the efficiency of decision making to the effectiveness of decision making [5]. To quote Peter Drucker, efficiency emphasizes doing things right, while effectiveness emphasizes doing right things. The decision support approach recognizes the following:

- (a) difficulty of communicating the utility of sophisticated Management Science/Operations Research techniques to practising managers.
- (b) lack of appreciation of the complexities of real world by the Management Scientists/Operations Researchers.

- (c) apprehension that an imposed optimal solution may put into question, the value of experience, maturity, judgement and wisdom gained by real world practice.

Recognizing these realities the DSS philosophy tries to support decision making rather than substitute decision makers through the use of models. It also lays considerable stress on the involvement of the decision maker directly in the decision making process, rather than his indirect involvement through the management scientists, which often is considered to be the essence of MIS approach.

Developments in two distinct areas within the last few years have made the DSS approach particularly appropriate [118]. First, there has been considerable technological progress - the evolution of computer equipment with substantial power at affordable costs, the technology of database, the personal computer revolution making computing power available at desktop, the communication revolution enabling remote access to data and resources, the emergence of interactive computing and the low cost availability of user-friendly software [118]. The second development in the last few years has been a conceptual one. An understanding of the inherent structure of information systems within organizations is emerging. Also, we are adding to our knowledge, the ways in which human beings solve problems and the ways in which we can build models that capture aspects of their decision making processes. These

insights provide us with some important concepts for system design [151].

The progress in these areas have been dramatic, and our planning and control systems should reflect the new capabilities. We now can build entirely new kinds of systems that dynamically involve the manager's judgement and support him with flexible access to data and models. These systems are known as Decision Support Systems.

1.2 OPTIMIZATION MODELLING:

Most people think of a model as being a small-scale version of a large object - a model ship or a model plane. The sense in which models are used in Management Science is analogous, but not quite the same. Ships can be scaled down with no material loss of realism in certain important characteristics. But business systems cannot be scaled down easily. We need a representation, which will behave similarly. A business contains legal, psychological, sociological, engineering and other factors and is a very complex affair. It cannot be represented in a scaled down form without serious oversimplifications. After prolonged experimentation with various sorts of representations, it has been found that a mathematical model is a useful kind of model of a business enterprise. The set of mathematical equations does not bear any physical resemblance to the business enterprise, of course, but the

equations can be chosen so that the values of variables relate to one another in a manner comparable to the relationships between variables in the organization. If the variables in the model have real counterparts, and if the relationships between the variables in the model correspond closely to the relationships in the real business system, we should be able to test the probable effect of changes in the real system by making equivalent changes in the model. Evidently, there is a great need for care in devising the system of equations so that the predicted outcomes are realistic.

The great advantage of Decision Support Systems is that the equations do not need to deal with all the factors at the same time and one can even omit some of them. The manager can supply his own estimate for some of the parameters based on his experience. Conversely, the manager can safely rely on his model to carry out the numerical and analytical parts of the process. The interaction of the manager with the model permits the strengths of the computer and the strengths of the manager to be brought to bear on the same problem at the same time. The intuitive judgement of the manager, with his years of experience of the business system cannot be computerized. Similarly, the data storage and retrieval capacity of the machine and the calculating power of the model using the machine are far beyond the capabilities of a manager. Each can contribute to the decision through a decision support system [118].

The use of models in management science is definitely not new. However, their application using the decision support approach is relatively new. In fact, this has led to a situation, where the discipline of modelling has advanced rather slowly while the disciplines concerning the analysis and development of techniques to solve such problems have grown much faster. It is imperative at this stage to redress this imbalance, as has been pointed out by Geoffrion [76] recently.

The first attempt at modelling in a supportive environment, came rather interestingly through the simple spreadsheet modelling [23,75,76]. This gave the practising manager, an opportunity to get involved in the modelling process directly. While the power of spreadsheets is definitely limited, the richness of its applications in real life has almost created a revolution [49]. In a parallel vein, modelling languages like IFPS [80,134] had been steadily in use in restricted areas of management like Financial Planning. What is needed now is to extend the scope of modelling systems that are directly used by managers, to include the capabilities of the sophisticated techniques of Operations Research/Management Science.

Among the several techniques of Operations Research/Management Science, optimization models using mathematical

programming have been the most extensively used tools. Mathematical programming in general and linear programming in particular have the following significant advantages:

- (a) ability to model a wide variety of applications
- (b) availability of tools/techniques to solve reasonably large problems that occur in real life situations.

The early applications of linear programming did evolve from simple numerical analysis software into sophisticated matrix generators. However, to embed optimization techniques within a decision support environment one needs to enlarge the scope of such software to the entire life cycle of modelling. This alone will allow a true interactive modelling capability rather than a mere solution capability. Such an approach will create a re-emergence of mathematical programming applications, improve modeller productivity, enhance the modeller-manager communication and ultimately to a better acceptance of the models [76]. The research reported in this thesis is primarily to design a support system for such a situation. Such an exercise, it is hoped, will lead to a new generation of decision support systems. Moreover the ideas developed would be useful for generating far more general decision support systems that may involve many of the techniques of Operations Research in an integrated manner. Geoffrion [76] has recently

pointed out the benefits of such a philosophy, which he terms a general modelling framework.

1.3 OUTLINE OF THE THESIS:

Chapter II contains the literature review. First the developments in the area of decision support are reviewed. This is followed by a review of the literature pertaining to problem processors, matrix generators and modelling languages for mathematical programming. The capabilities of a number of ~~the~~ existing matrix generators is contrasted with the capabilities of a number of emerging modelling languages. The need for a model management system with enhanced capabilities of data management is established.

Chapter III outlines the research goals of our modelling system LAMP alongwith the possible benefits arising out of research efforts in this direction.

Chapter IV covers the various features of the modelling language proposed in this thesis. The syntax of the language is specified in the form of an EBNF Grammar [98]. Next follows, the features of the language, its capabilities and the implementation details. A sample problem illustrating the details is also provided.

In Chapter V the need for an interface between the modelling language(s) and problem processor(s) is established.

A standard in the form of a set of relations to represent the data associated with arbitrary linear programs is proposed.

Chapter VI demonstrates the use of a standard relational database management and a query language to accomplish the model manipulation functions. A wide variety of analysis and report writing capabilities that is typically needed in most modelling situations is accomplished using the query language.

Chapter VII summarizes the conclusions and lists the recommendations for further research work in this area.

CHAPTER II

LITERATURE SURVEY

2.1 INTRODUCTION:

Decision Support Systems (DSS) represent a broad based spectrum of ideas that support the executive decision processes with flexible access to data and models [118]. The DSS approach emphasizes the fact that management involves primarily decision making and managers need a direct support, through tools and techniques in the decision making activity. The revolution in computing power in general and micro-computing in particular has helped in translating this support philosophy into real life applications by involvement of the decision maker in the modelling processes [22]. In fact decision support systems and associated modelling represent a renewed interest in Operations Research/Management Science (OR/MS) techniques [32].

It is generally observed that optimization represents a major application technique in the entire gamut of OR/MS techniques. Several studies have established this observation [121]. Linear Programming (LP) has enjoyed a widespread acceptance, due to ease of use, that is characteristic of linearity of constraints and objective function. Consequently, linear programs involving several thousand of variables and

several hundred of constraints have been traditionally solved in such diverse industries like Petroleum, Food, Transportation, Energy and Service Sector [176]. Almost all application of LP made extensive use of computational facilities leading to several software products called LP Packages. Applications of Nonlinear Programming (NLP) to real world problems are emerging only recently. Such systems that lend computational support to the solution of mathematical programming problems are usually referred to as Problem Processors. They have evolved considerably in terms of power and complexity in the last four decades. The early LP Packages primarily exploited the numerical problem solving capability of the digital computer [75]. Undoubtedly, but for the availability of a digital computer there was no possibility of solution. Yet, the numerical analysis is only a small part of the overall solution process. Modellers soon started looking for support in non-numeric computation as well, leading to the early matrix generators.

Linear programs are expressed in one kind of form for human designers or modellers, but in a quite different form for computer algorithms. Thus the translation from modeller's form to the algorithmic form is an essential task. Traditionally, this task of translation has been divided between human and computer through the writing of computer programs known as matrix generators (MG)^[68]. A matrix generator is a computer program that writes out the algorithmic form i.e., the

coefficient matrix of a linear program. First, a person reads the modeller's form and writes a matrix generator program for it. Then the matrix generator program is executed to produce an algorithmic form. Very soon they were found to be inadequate and often artificial [68]. As a natural consequence this led to the growth of modelling language systems as an alternative approach.

A modelling language system leaves almost all the work of translation to the computer. The key concept to such an approach is a computer readable modelling language that expresses a linear program in the same manner that a modeller does. Further, it is observed that modelling languages lead to a more reliable application of linear programming at a lower overall cost with reference to verification, modification, documentation, independence and simplicity aspects [68]. Modelling languages also provide a flexible access to modelling by providing support in all four stages of modelling viz. formulation, data analysis, numerical solution and analysis of results.

Many of the real life models using linear programming need large volumes of data to be processed as part of the solution process. The job of managing this large volume data is by no means a trivial job. Efforts to integrate data management functions have been attempted elsewhere in an independent manner. With the tremendous growth in database technology in

the recent years we feel that the integration of database techniques will go a long way in building powerful model management systems. This will be reflected in the philosophy of our approach outlined later.

The evolution through the matrix generators, model management tools and the recent intelligent systems for mathematical programming [83] is indeed impressive. We take stock of these developments in the subsequent sections, which provide the background for our work. Our work proposes an architecture for the next generation of problem processors, that build on the strength of the current problem processors. A representative system incorporating this architecture is also built, to demonstrate the power of the designs using this architecture, which is the major thrust of this thesis.

2.2 PROBLEM PROCESSORS

Solving even small mathematical programming problems demands a prohibitive amount of computational effort, if undertaken manually. Thus all practical applications of mathematical programming require the use of computers. In this section we examine the role of the digital computers in this task. The task of solving linear programs is greatly facilitated by the very efficient linear programming codes that computer manufacturers provide for their computer models. The representative ones being IBM's MPSX [92], CDC'S APEX [47], Burrough's MODELER [33], Honeywell's MPS [89] etc. Recently independent

softwares like LINDO [145], GINO [114], GRG [111], MINOS [128] have come into being. In a matter of one or two days, a potential user can become familiar with the program instructions that have to be followed, in order to solve a given problem using a specific commercial code. The ample availability of computer codes of good quality is one of the basic reasons for the increasing impact of mathematical programming on management decision making. Many of the codes have an extended capability to solve integer and limited nonlinear programming problems (separable problems, linearly constrained nonlinear optimization problems, etc.) as well. From simple, rigid computer programs of the yester-years, the modern linear programming codes have evolved into sophisticated systems with modules that handle input information, provide powerful computational algorithms and report the model results. Most of these advances have been made possible by developments in software and hardware technology as well as break-throughs in mathematical programming theory. The continual improvement in computational capabilities of mathematical programming codes is in reducing running times and computational costs, to facilitate solutions of larger problems and to extend the capabilities to more general non linear, discrete optimization. In the next sub-section we will take stock of the basic features common to all good current-day systems emphasizing the important concepts.

2.2.1 Basic Features

The basic features can be broadly classified into:

- a) Input Specifications
- b) Solution Techniques
- c) Output Specifications

a) Input Specifications:

The first task when solving a linear programming model using a computer code is the specification of model structure and the input of the corresponding values of the parameters. The important elements, the user has to specify are:

- . Number of the decision variables, constraint(s), objective function(s), right hand side(s)
- . Numerical values of the cost coefficients, matrix coefficients and right hand sides
- . Nature of the constraint relationship
- . Free and bounded variables
- . Nature of the optimization
- . Title of the problem.

To facilitate easy and meaningful analysis/retrieval, symbolic identification of variables and constraints through names (typically of eight characters length) are provided. Typically only non-zero coefficients need to be input. Several small systems allow free-form input e.g., MPOS [46]. Many of the codes have evolved over the years, a standard called MPS format, originally used by IBM's MPSX [92]. This standard

has become archaic and we will propose an alternate elegant relational format, later in this thesis.

Input specifications also include several summary results of the data to help the user, undertake checks for consistency; typical examples being:

- . Row, column count
- . Matrix density (percentage of nonzero coefficients)
- . No. of nonzero elements in each row and each column
- . Values of the smallest and largest element in the coefficient matrix, right hand side (s), objective function (s)
- . Print-out of equations in expanded form
- . Picture of the full coefficient matrix or a condensed form of the coefficient matrix.

Sometimes, input data need to be scaled. The simultaneous presence of both large and small numbers in the matrix of coefficients should be avoided whenever possible because it tends to create problems of numerical instability. This typically can be avoided by changing the unit of measure of a given variable, say from kg to 1000's of kg.

Later in this chapter we will see further evolution of input specifications into full fledged software systems known as matrix generators.

b) Solution Techniques:

There has been considerable concern in the development of problem processors to reduce the computational time required to solve programming problems, by improving the techniques used in the optimization stage of the problem. This has been accomplished primarily by refinements in the matrix-inversion procedures using triangularization; representation using sparse matrix technology; path selection criterion by allowing multiple pricing of several columns to be considered as possible incoming variables; use of more efficient pre-solution techniques by providing crashing options to be used in order to obtain an initial feasible solution or by permitting the user to specify initial basis. To avoid the presence and/or growth of numerical errors, most codes use original values of the coefficient matrix to reinvert the basis periodically. This process maintains accuracy. The frequency of reinversion can be user specified. Also the user is allowed to specify feasibility tolerances that are designed to check whether or not the values of non-basic variables and the reduced cost of the basic variables are zero. There are also pivot rejection tolerances that prevent coefficient very close to zero from becoming a pivot element in a simplex iteration. If necessary the systems allow high precision arithmetic (generally double precision) to be specified by the user to decrease the computational errors. In essence, the systems allow control of the solution process by the user in a detailed way.

c) Output Specifications:

Typical output information provided by standard commercial codes includes:

- . Optimum value of the objective function and optimal values of the decision variables
- . Slacks/Surpluses in the constraints
- . Shadow prices for the constraints
- . Reduced costs for the decision variables
- . Ranges on the coefficients of the right hand side
- . Ranges on the cost coefficients
- . Value transitions resulting from changes in right hand side/cost coefficients
- . Sensitivity and Parametric Analysis.

Occasionally some of the codes provide summary results for reporting to management directly. Others provide a report writer program that can be user written and appended to the system for report generation. We will review some of the capabilities in our discussion of matrix generators in the next section.

2.2.2 Current Trends:

The current trends in the evolution of problem processor include development of model management system (discussed later in Section 2.4); improved sophistication in their ability to solve nonlinear and integer programming problem, reviewed in [175]; porting the software to micro computers resulting

in a rich collection of micro based software for mathematical program, reviewed in [148, 149], adding graphic support, reviewed in [175], adding explanation capability to linear programming packages [83]. Spurred by the spread-sheet revolution [23], planning languages are being increasingly used by modellers. Extending optimization capabilities to planning languages is being explored rather recently. IFPS/Optimum [80], VINO [49] represent two philosophies in this direction.

2.3 MATRIX GENERATORS:

As mentioned previously, a matrix generator is a computer program that writes out the algorithmic form i.e., the coefficient matrix of a linear program. A number of matrix generator and report writer systems which we term as matrix generators essentially use a standard mathematical programming processor. These systems are quite significant for any organization that uses mathematical programming (MP) as a serious modelling tool. These systems include matrix-generator (MG) programs and report-writer (RW) programs which may be written in high-level computer programming languages viz. FORTRAN, ALGOL, PL/1 or in a special-purpose matrix generation and report writing language. In such a system, first the matrix-generator program reads and processes the "PROBLEM DATA" presented in the form of sets of tabulated data to produce a card image of INPUT FILE; then the solver or optimizer generates solution values; finally the

report-writer program extracts relevant information and presents in a desired format. The card image "INPUT FILE" contains row names that are coded by suitable text expressions; column names that are also coded by appropriate text expressions; the coefficients of the problem matrix; the RHS (right hand side), BOUNDS and RANGES information; some information concerning starting basis. The report-writer program generates the specific information from the solution values obtained by the optimizer and presents the same information in a suitably tabulated format. The report-writer program usually refers the "PROBLEM DATA" from the "INPUT FILE" and may also carry out some arithmetic operations on the solution values to generate desired information.

The most prominent among a number of matrix generator and report writer systems are:

MODELER (Model Development Language and Report Writer)

- . developed in 1980 by Burroughs Corporation [33].

MaGen (Matrix Generator)

- . developed in 1977 by Haverly systems Inc. [86].

MGRW (Matrix Generator and Report Writer)

- . developed in 1972 by IBM World Trade Corporation [91].

GAMMA (Version 3.4)

- . developed in 1977 by Sperry Univac Computer System [154].

DATAFORM (MPS-III Version)

- . developed in 1975 by Ketron Inc. [108].

APEX (Version - III)

- developed in 1979 by Control Data Corporation [47].

These systems commonly incorporate at least the following features:

- input of problem data in a tabular form
- construction of row and column names by name expression
- use of constants or arithmetic expressions to specify the matrix, RHS, BOUND, etc., coefficients by suitable row or column generator clause (procedure)
- access to the solution file to obtain solution values, reduced costs, ranges, etc.
- formatted and printed tabular information.

Current matrix generators incorporate their own MG languages in which the programs are written. They commonly presume a sequence of four forms: modeller's form, MG form, Standard (MPS) algorithmic form, and specialized algorithmic form. These languages incorporate loops, assignments, transfer of control or other executable statements in describing the computer program.

Matrix generation is certainly a great advance over translation by human labour alone, particularly for very large linear programs. However, they suffer from potential difficulties viz. verifiability, modifiability and documentability which are aggravated by introduction of matrix-generator form. [68]. The associated drawbacks of matrix generators are:

Verifiability: This being the major consideration in a linear program, refers to determining whether modeller's form correctly represents the reality and also determining whether the algorithmic form is a correct translation of the modeller's form. The verification of matrix-generator program necessitates program debugging at the cost of human time and computer time.

Modifiability: This relates to determination of new modeller's form corresponding to any change in real life problem situation and translating the new modeller's form to a new algorithmic form. Thus in the case of matrix generators, any modification essentially amounts to revising matrix-generator computer program which again aggravates the task of program debugging.

Documentability: This relates to maintaining of all relevant information pertaining to different forms and their relationship. The matrix generator requires additionally, documentation of matrix-generator program which is fairly lengthy, time consuming and unpopular with programmers; yet inadequate documentation can eventually make it difficult to use or change a program.

The matrix generators also have drawbacks related to the following areas:

Independence: The weakness of matrix generators lies in their involvement with the algorithmic form. The conceptual binding with particular algorithmic form imposes restrictions with regard to features viz. naming of LP components, ordering of coefficients and representation of special constraints. This in turn necessitates modeller's familiarity with particular algorithmic form which could be avoided only at some additional cost.

Simplicity: The final drawback of matrix generation stems from the basic approach of translation task i.e., introduction of intermediate MG form. This additional form increases the modeller's burden and complicates the conceptually simple job of translation of modeller's form to algorithmic form.

Further, the majority of matrix generators are more influenced by the algorithmic form i.e., MPS form. Therefore, the modellers using these matrix generators have to follow various conventions and restrictions of matrix-generator data format and of MPS form at the cost of extra work and complication. These above mentioned difficulties lead to the development of modelling languages described in next section[68].

2.4 MODELLING LANGUAGES :

Modelling Languages (ML) as mentioned earlier are natural evolution from the matrix generators. They are designed to provide a superior support in all the phases of modelling. In this system, the modeller first creates a symbolic modelling language description of the mathematical program of interest and then a computer system reads the above symbolic description along with the corresponding explicit set and parameter data to translate it to the desired algorithmic form. Thus a modelling language system serves to divide the work of translation between a person and computer with greater portion of work shared by computer as compared to matrix generators. ML-like systems for linear programming have mostly been developed since late seventies. The more sophisticated of these languages incorporate some kind of symbolic indexing, in order to conveniently represent fairly large models. The major systems employing modelling languages' features are ALPS [170], GAMS [105], LMC [124], LP MODEL [103], MAGIC [52], MGG [147], UIMP [167], XML [68], SM [75]. They are further outlined below.

ALPS (Advanced Linear Programming System):

ALPS is an easy to use mathematical programming package developed in 1977 by United Computing Systems, Inc., Washington, USA. The modelling language uses "FOR" loop statements to

define constraints. The language notation is fully algebraic and it incorporates general linear expressions with limited variety of indexed sums. The data description is based on symbolic parameters only and subscripted identifiers are used for model component names. It allows for coefficients to be given in a row-wise organization. Special constraints such as simple bounds can be described like other constraints. The model description is independent of explicit data. Further details regarding major features are available in [158].

GAMS (General Algebraic Modelling System):

A general algebraic modelling language package was developed at the World Bank Development Research Center, Washington, USA. It has been in extensive use since 1979. The modelling language uses assignment statements. The language notation is algebraic with some exceptions and it incorporates general linear expressions and indexed sums. The data description is based on symbolic sets and parameters and subscripted identifiers are used for model component names. It allows for coefficients to be given in a row-wise organization. Special constraints such as simple bounds can be described in a special manner. The model description is not independent of explicit data. Further details regarding major features are available in [121].

LMC (Linear Modelling Capability):

A subsystem of Conversational Modelling Language (CML) was developed at Yale University Institute for Social and Policy Studies Center, Connecticut, USA. It has been in use since 1977. The modelling language uses assignment statements, The language notations are English like rather than algebraic and it incorporates general linear expressions and indexed sums. The data description is based on symbolic sets and parameters and subscripted identifiers are used for model component names. It allows for coefficients to be given in a row-wise organization. Special constraints such as simple bounds can be described like other constraints. The model description is independent of explicit data. Further details regarding major features can be found in [124].

LPMODEL (Linear Programming Modelling Language):

A modelling language system for linear programs was developed at IBM Israel Scientific Center, Haifa, Israel. It has been in use since 1980. The language notation is algebraic with moderate alteration (e.g., "SUM" for summation notation) and it incorporates general linear expressions and limited indexed sums. The data description is based on symbolic sets and parameters implicitly defined and concatenated identifiers are used for model component names. It allows for coefficients to be given in a row-wise organization.

The model description is independent of explicit data. Further details regarding major features are available in [103].

MAGIC (Matrix Generator Instruction Convertor):

A modelling language package was developed at University of Edinburg, Scotland. It has been in use since 1982. The modelling language uses "FOR" loop statements to define constraints. The language notation is fully algebraic and it incorporates limited linear expressions and general indexed sums. The data description is based on symbolic parameters only and subscripted identifiers are used for model component names. It allows for coefficients to be given in a row-wise organization. Special constraints such as simple bounds can be described like other constraints and ranges can be described in a special manner. The model description is not independent of explicit data. Further details regarding major features of this system are available in [52].

MGG (Matrix Generator Generator):

A modelling language package was developed by Scicon Computer Services Ltd., England. It has been in use since 1975. The language notation is fully algebraic and it incorporates very limited linear expressions and indexed sums. The data description is based on symbolic sets and parameters and subscripted identifiers are used for model component names. It allows for coefficients to be given in a row-wise organization. Special constraints such as simple bounds and

generalized upper bounds along with ranges can be described in a special manner. The model description is independent of explicit data. Further details regarding major features can be found in [146].

UIMP (User Interface for Mathematical Programs):

A mathematical programming package was developed by UNICOM Consultants Ltd., Surrey, England. It has been in use since 1976. The modelling language uses loop, assignment statements and transfer of control to define constraints. The language notation is fully algebraic and it incorporates very limited linear expressions and general indexed sums. The data description is based on symbolic sets and parameters and subscripted identifiers are used for model component names. It allows for coefficients to be given in a row-wise or column wise organization. Special constraints such as simple bounds and ranges can be described in a special manner. The model parameters may be independent of explicit data. Further details regarding major features are available in [61],[68]

XML (A Prototype Modelling Language):

XML is a hypothetical modelling language based on the sort of modeller's form. It has been developed by Fourer at MIT, Cambridge, USA. The modelling language system has five major sections: sets, parameters, variables, objective and constraints. The language notation is algebraic with moderate

alteration and it incorporates both indexing and arithmetic expression. The data description is based on symbolic sets and parameters and subscripted identifiers are used for model component names. It allows for coefficients to be given in a row-wise organization. It is syntactically and semantically simpler. A detailed specification of the modelling language system is available as technical report by Fourer [67],[68].

SM (Structured Modeling):

SM is being developed at Western Management Science Institute, University of California, Los Angeles, USA. It endeavours to provide a formal mathematical frame work, language and computer based environment for conceiving, representing and manipulating a wide variety of models. The frame work uses attributed acyclic graphs, set partitions and hierarchies to represent the semantic as well as mathematical structure of a model. The language is computer executable and yet natural for novice users. The computer based environment is evolving. Further details regarding salient features can be found in [76].

Other systems like LINDO [145], MPOS [46] do have the flavour of modelling language even though they are primarily problem processing systems.

Critical examination of the modelling systems reveals the following: all of them use modelling languages that

resemble modeller's form to some degree and do not involve programming, though traces of programming languages forms can be found especially in ALPS and UIMP. UIMP and MGG use MPS naming schemes i.e., eight characters-long names for their entities whereas other systems do not have such restrictions. All of them permit a constraint-wise (row-wise) organization, except UIMP, which allows column-wise organization also. All except LPMODEL use subscripted identifiers for naming model components; LPMODEL uses concatenated identifiers of unrestricted length for the similar purpose. ALPS permits quadratic objective. MAGIC, MGG, UIMP and ALPS support discrete (integer) optimization model as well. ALPS, MGG and UIMP are available as commercial products[68].

As a group, the greatest strength of these systems is their ability to raise the productivity of optimization applications. The serious weakness of most of these systems is that they are wedded to one particular modelling paradigm and lack generality to encompass most of the other modelling paradigms developed in OR/MS. They also lack integrated facilities for query and immediate expression evaluation. The above observations match with the aims of structured modelling (SM) and suggest for the need of an appropriate system providing synthesis of model management, data management and associated interface, which is the focus of this thesis.

2.5 DATA MANAGEMENT:

2.5.1 Introduction:

Integrating several sets of data needed by several applications and sharing the data resource for simultaneous access by several authorized users has been the dream of information scientists. This has been made possible with the technological growth of computer storage devices capable of storing large volumes of data at a lesser cost, in the recent years. This is generally termed the technology of database. A modern definition of database is a set of structured data stored on the computer accessible media that satisfies the needs of several users simultaneously, in a selective manner yet in a reasonable time frame. In order to effectively manage such a database, a sophisticated software is generally needed. Such a software is generally termed a database management system (DBMS). Basically a DBMS provides the means of organizing data on peripheral storage devices and retrieval procedures necessary for internal processing of the stored data. The user first describes the structure of the data and the tasks in abstract terms and leaves the job of determining the best operating mode to the database management system. The database structure is described at the conceptual level by a logical schema which provides a view of the data to the user. At the internal level the structure is specified by a physical schema. Usually the

structure is specified to the DBMS using a data definition language (DDL); the manipulation of the data stored in the database is specified through a data manipulation language (DML). The foundation for the structure, however, is the notion of a data model described in the next section.

2.5.2 Data Model:

The data model is a structured tool used to understand and interpret the real world. In order to facilitate communication, it is useful to regroup the real world objects with which we are interested, into homogeneous classes. A data model allows one to define relationships which may exist between different classes of objects. Reflecting different views of the relationships amongst the classes of objects several data models have been proposed in literature. Of these, there are three prominent data models viz. the hierarchical model, the network model and the relational model which evolved over the last decade. Using hierarchical approach, conceptual model is viewed as a tree graph where nodes represent classes of objects and branches represent association between two classes. Hierarchical graphs have a root and the nodes emanating from root are labelled as sons, grandsons, etc. of the root. Using the network approach, the conceptual model is viewed as an ordinary graph where nodes represent classes of objects and arcs represent association between two classes. The network model differs from

hierarchical model in that there is no hierarchical limitations on association. An introductory description of major data models is available in DBMS texts [50,164 , 166]. The widely used hierarchical model based system has been the IBM's IMS (Information Management Systems). The CODASYL (Conference on Data Systems Languages) Committee's recommendations were based on a Network Model. Consequently a number of implementations based on network approach were made available in the last decade by several main frame/mini-computer manufacturers including IDS [40] from General Electric, DBMS-10 [57] from Digital Equipment Corporation. The growing importance of the relational model in the recent past suggests that it is likely that the models in the future will be predominantly relational (due to reasons elaborated in the next section).

2.5.3 Relational Data Model:

The relational data model views the logical schema as tables having rows and columns, where each row represents a record and columns represent various attributes (fields) related with the object. The tabular arrangement of data is an elegant representation. This model is based on the feature of a mathematical relation which is a set of tuples having tuple elements as attributes of the object. The mathematical foundation of relational data models permits elegant and concise definition and deduction of their

properties [38]. The relational concept was first applied through AOV (attribute, object, value) theory along with an algebraic theory of data proposed by Childs [39], about 1968. Later around 1970 Codd [43] proposed the relational data model concept based on ^{the} notion of relation. Being conceptually simple, the basic relational model had a tremendous impact on database management in general and on data models in particular. Because of the elegance of the structure and the mathematical foundations of the theory of relations, the relational model has become the main stay of research and development in the database area. The relational data model is claimed to be superior to other data models. However, subjective this observation be, the relational approach has definitely achieved a great goal; ~~of~~ bringing together the practitioners who implement systems and researchers who conceptualize them. The various prototype and commercial DBMS that implement relational approach include System R [9], INGRES [162]. Consequently we concentrate only on relational data model in our thesis.

2.5.4 Query Languages.

Data Manipulation Languages (DML) that allow manipulation of data stored in database are generally known as query languages. They are non-procedural compared to the procedural nature of conventional programming languages like PASCAL and FORTRAN. They may have features of ^{common} ~~basic~~ programming languages

such as assignments, iterative loops and conditional statements permitting one to refer data as well as to specify some manipulation functions such as insertion, deletion, updation and query. Since relational data models are based on relations (i.e., set oriented concept), they naturally lend themselves to specification operations. The query languages use specification operations which may be binary (e.g., sum, product, difference and join) or unary (e.g., selection, projection). The salient feature of the query languages is their ability to define a new relation based on the existing relations using relational algebra or similar (relational calculus) type of operations. The new relation is usually made available for use as a picture (snapshot) or as a view which is an extension of schema. These query languages can be classified as:

- (i) Predicate Calculus Language having origin in relational calculus concept and based on assertions over a set of tuples of a relation (e.g., QUEL [87], ALPHA [44]).
- (ii) Display Oriented Language where user is required to fill in blanks on a CRT display to formulate a query (e.g. QBE [180]).
- (iii) Algebraic Language or Mapping Oriented Language which is based on operations having relations as operands (e.g., SQL [55]).

Further, the query language enables the desired information to be extracted in a relational format by successive application of binary operators (e.g., sum, product, cartesian product, difference and join) or unary operators (e.g., complement, selection and projection) on stored database relations, treating them as operands. Various query languages developed using relational data model concept are QUEL (INGRES) [87], QBE [180], SEQUEL [29], SQL [95]. They are briefly outlined below.

QUEL, a language based on relational calculus was developed for INGRES relational system at University of California, Berkeley. QBE (Query-By-Example), a display oriented query language defined by Zloof [180] which runs on INGRES.

SEQUEL (Structured English Query Language) is the modification of language SQUARE developed by BOYCE. The modified language was developed for System R.

SQL (Structured Query Language) is the modified version of SEQUEL. It is an English-keyword, set-oriented relational data language. It has been implemented in System R DBMS. It is based on the concept of mapping operation.

2.6 CONCLUSIONS:

The survey of literature does indicate the desirability of building model management systems that integrate data management aspects as well. Such systems alongwith a problem processor will provide support to a modeller in the entire life cycle of the modelling process, and not just in the numerical solution of the problem. In our view, numerical solution of the linear program constitute but a small, though significant, part of the modelling activity. Hence the widening of the support to the model builder is likely to be of major significance in the modelling research. The work reported in this dissertation develops this thesis further in the form of an architecture for modelling support and a representative system using this architecture.

CHAPTER III

MODELLING SYSTEM LAMP

3.1 INTRODUCTION:

Optimization modelling has witnessed an impressive growth and evolution, as indicated by the literature surveyed in the last chapter. In spite of such advances, optimization has not received a widespread acceptance, which it rightly deserves. Possibly it is due to some of the problems listed below, as identified by Geoffrion in [76]. These suggest some desirable features for a new generation of optimization modelling systems, which is the major thrust of this thesis.

One of the major problems in the acceptance of optimization modelling is the low productivity as perceived by the users. It is contributed by different representations - a natural representation suitable for managerial communication, a modeller's form for the analytic use by the OR/MS professional and an algorithmic form that is computer executable. Such multiple representations demand too many different skills to complete even small projects. Matrix generators and the modelling languages partly solve this problem by providing convenient automatic translation from one form to the other. Another factor

contributing to low productivity is the labourious task of interfacing models with problem solvers. The commonly used MPS Standard [92] for linear programming is ancient and is not suitable for modern systems. Finally the available modelling systems cater just to one or two phases of the total life cycle of modelling such as input data translation and algorithmic solution. The phases left out are - requirement design, testing, revising, maintaining, documenting, explaining, analyzing, reporting and evolving the models. Typically such low productivity leads to poor managerial acceptability. Luckily a number of new opportunities, that are being created are potentially useful in mitigating many of the problems mentioned above. The prominent among them are:

- (a) Desk top computing revolution - this offers a plethora of possibilities for optimization modelling in a more productive manner and in ways that are acceptable to managers.
- (b) Progress in database systems -- data management is an important activity in most optimization models as most real life applications tend to be large. Managing such large data manually is a hopelessly difficult job.
- (c) Popularity of spreadsheet modelling - this had enabled the managers to get directly involved in modelling, may be inadvertently as pointed out by Bodily [23], thereby removing the first block in acceptability of models by the managers.

The problems and opportunities enumerated above suggest that the evolution of problem processors through matrix generators and modelling languages must go several steps further in the form of new generation of modelling systems with the following features:

- (a) independence of model representation and model solution with general interface standards to facilitate the integration of a variety of solvers.
- (b) extending the scope of the computational support to the modeller in all aspects of modelling.
- (c) integrated data management and adhoc query facilities.

Keeping in view the points cited above we decided on a set of goals for our research, outlined in the next section.

3.2 RESEARCH GOALS:

The desirable features of a modelling system, as outlined earlier, represent a possible architecture for a new generation of systems that are likely to influence optimization modelling in a significant manner. In order to demonstrate our thesis we will set out to build an experimental system, hereinafter referred to as LAMP, incorporating this architecture and illustrate through detailed examples the advantages that are likely to accrue arising out of such a point of view. No attempt is being made to develop a full fledged software of the kind that is readily usable. We feel that such an effort

calls for an altogether different effort and skill and may even be premature at this stage. We need a translation system i.e., a modelling language subsystem to translate the modeller's form into an algorithmic form. The modeller's form must be so chosen that it is communicable to the people, not necessarily to OR/MS professional, yet easy enough for translation. Most of the modelling languages like GAMS [121], LPMODEL [103], XML [66] are still evolving at this stage and aim at developing them into powerful and independent systems of commercial/professional quality software. Since our purpose is modest and purely of an experimental interest, we plan to build a system that is small, elegant and yet of reasonable power. Having built a modelling language we would like to interface it with an independent solver LINDO [145] that has been in use at Indian Institute of Technology, Kanpur for the past few years with a good amount of reliability. We do need an interface standard. We are not happy with MPS Standard [92] which is not well suited for query and integration with data management. It appears that an alternate view using simple normalized relations, of the input and output data of a linear program would simultaneously meet the twin purpose of an elegant interface standard as well as a convenient link to the data management function. To extend the scope of the system to include data management function, we feel that it will be advantageous to use the

well developed technology of data management and query languages to do the entire "what-if" analysis and report writing. So we plan to interface our system to an elegant, yet powerful DBMS software, viz. UNIFY [168] using yet another query language SQL [95] which is one of the popular members of its kind. Another simple reason for using this system is its availability at IIT Kanpur. Effort will be taken to ensure that the concepts being demonstrated are independent of the products that are likely to be used i.e., LINDO, UNIFY or SQL, so that they remain sufficiently general. Ultimately all the units would have been integrated so that the support provided, enables the modeller to conduct a truly interactive modelling and that the support spans the entire life cycle of the modelling process. To summarize, the research goals are:

- (a) Develop a simple yet powerful modelling language that translates the modeller's form into an algorithmic form.
- (b) Develop an interface standard between model language system and problem processor system (an alternate to MPS System [92]).
- (c) Incorporate data management capabilities using the ideas of relational data management.
- (d) Study the use of general purpose query languages to edit, modify and conduct "what-if" analysis of the models.

- (e) To maintain a modularity of the system so that they can be evolvable, communicable, maintainable and documentable.

3.3 RESEARCH BENEFITS:

We foresee a tremendous demand for a new generation of modelling systems with the architecture outlined earlier. Demonstrating a system built on such architecture is likely to vindicate our viewpoint that such architecture has attractive advantages over the current generation of systems. Such a viewpoint is likely to overcome the major problem of poor acceptance by user. Specifically, the research tasks mentioned earlier are likely to lead to the following benefits.

Task (a) outlines yet another modelling language which in no way can be claimed to be very superior to many of the experimental languages like LPMODEL [103], GAMS [105], XML [66]. It can nevertheless be said that none of the experimental ones have a fully satisfactory representation of modellers form even with respect to linear programming. What is ultimately needed is a language that can represent far more general models. So any addition can only be welcome at this stage. What is significant is that a more general structure of the modelling language would evolve with many more such experiments. In future it is anticipated that with the availability of program generators [100], generalized parsers [2], compiler-

compilers, experimenting with newer languages is likely to be less expensive. It may be noted here that the modelling language being developed would take advantage of the fact that many of the data that goes into optimization models in fact comes from corporate databases. These are by themselves expected to be relational in future and the future modelling languages would significantly benefit from such an observation.

Task (b) would enable the unbundling of the problem processor from the modelling languages, matrix generators and the report writers. In fact many of the modelling systems are built around specific problem solvers and such unbundling would lead to a happy marriage of powerful problem processors and user friendly modelling languages.

Task (c) would greatly facilitate in the job of having to muddle through the volume of data that is typical of most optimization models. Use of relational data management provides the dual advantages of using the ideas of the well developed technology of database as well as an ability to maintain an independence between problem processor, modelling language and data management system.

Task (d) would herald an era of trully interactive modelling which would significantly affect both the modelling process and its ultimate acceptance by the user.

Task (e) would provide for an evolution capability of the model management system. It is definitely likely that each of the sub modules would undergo significant advances in the coming years due to the developments in conceptual mathematics, program generators, query languages, visual editors and the user interfaces. We have already outlined the possible impact on modelling language design due to developments in the area of program-generation. Developments in conceptual mathematics would improve the quality of problem processors. The emerging ellipsoid algorithm [84] and Karmarkar's implementation [101] may totally change the way linear programs will be solved in future. The future query languages would enable a far superior "what-if" analysis than what is possible today. Advances in visual editors and other graphic interfaces may give the modeller an unprecedented interactive modelling capability. Possibly in future an intelligent system providing explanation capability could be integrated with the system. In essence, the modularity enables the user an option of combining the advantages of developments in each of the areas without tying him down to any particular system.

CHAPTER IV

MODELLING LANGUAGE FEATURES OF LAMP

4.1 INTRODUCTION:

The previous chapter has outlined the broad objectives in the development of a modelling system LAMP at IIT, Kanpur. As set out in the earlier chapters the first task is to build a modelling language that translates the modeller's form into an algorithmic form. We first take stock of the relative strengths and weaknesses of the earlier systems outlined in the literature review. These help us to form our design goals laid out in the next section. The following sections elaborate the features of LAMP as implemented and its grammar. A detailed description of the capability of LAMP in its job of translation is included in Section 4.3.

Many real life optimization problems can be modelled as linear programming problems. In these situations a modeller describes the LP in a symbolic and readable form using familiar algebraic notation for variables, constraints and objective. However, for further processing, the computer algorithm requires the model in explicit and computer readable form. These two forms of linear program viz. the modeller's form and

the algorithmic form, are not much alike and both of these forms are essentially required for optimization, as no single means of expression could serve as both the forms. Thus any real life application of linear optimization requires translation from one form to the other form. Translation from the modeller's form to algorithmic form is thus an unavoidable task in mathematical programming based models. This process of translation has been found to be quite expensive, time consuming and difficult on the part of the modeller who plans to model real life optimization problems^[68]. Traditionally this job of translation is accomplished by a matrix generator software.

In the traditional approach to translation, the modeller first converts a symbolic modeller's form to a special computer program. The execution of that computer program generates a linear program in the desired algorithmic form. The intermediate computer program employs specialized programming language and is usually referred to as matrix-generator program. Matrix generators offer numerous advantages compared to the manual translation of modeller's form to algorithmic form. The principal advantages are [68]:

- They provide an elegant way of organizing and storing problem data which is reasonably large in most real life modelling situations
- They enforce a uniform scheme for naming rows and columns

- . They enable the user to modify model structure or model data with much less work.

However, the common design of matrix generators had serious limitations:

- . Matrix generators run in batch mode and require their own file structure or environment
- . The algebraic form of model still has to be translated by hand to an MG program. Writing this program is difficult as it requires strong skill of special programming language
- . Large MG programs are hard to follow
- . Devising unique names for rows and columns is often awkward
- . They are not readily adapted to nonlinear programming situations, because most general nonlinear problems could not be expressed in terms of matrix columns.

In the modern approach to translation the idea is to have maximum possible work of translation done by machine itself i.e., the modeller should deal with the computer directly in modeller's form using a special kind of one such form. This special modeller's form should be easy to write and understand for the user and easy to read and translate for the computer. Central to such an approach is a machine readable modelling language that expresses the mathematical program in much the same way as the modeller does i.e., by employing a variant of algebraic form that is designed to be read by a computer system. This machine readable algebraic form is referred to as a modelling language (ML). It differs from common algebraic form mainly in employing

standardized notation and terminology. A modelling language may be treated as a modern replacement for traditional matrix-generator language. Essentially both are the ways of describing a linear program to a computer system. A modelling language makes no reference to the LP matrix. It serves only to represent an algebraic form of the model [66] whereas a matrix-generator language describes a linear program by specifying all of its non-zero matrix coefficients. The modelling language is not a programming language, where as a matrix-generator language is.

Consequently, a modelling language can make the LP modeller's life easier in a number of ways:

- . It is easier to learn than special purpose matrix-generator language.
- . The algebraic description of modelling language can serve as a documentation of the model.
- . It can identify constraints and variables by familiar method of subscripting.
- . It can be easily adapted to nonlinear models.

The concept of modelling languages has been successfully used for other kinds of modelling as well. However, most of the applications of modelling language to linear programming have been of smaller scale or have used the concept of a modelling language to a limited extent. Further, it is generally accepted that modelling languages lead to a more reliable modelling scheme often at a lower overall cost [68].

The modern approach of modelling languages has specific inherent advantages over traditional approach of matrix generators. The general drawbacks of matrix generators were mentioned in Chapter II. The advantages of modelling languages over matrix generators are outlined below (in the same order).

Verifiability:

The major consideration here, is to determine whether the modeller's form correctly represents the reality; one should also check whether the algorithmic form is a correct translation of the modeller's form. Use of modelling language necessitates proper conception of reality and its representation which can be treated as debugging of ML representation. This is relatively much easier as verification task reduces to an item-by-item comparison between modeller's conception and the possible realization from the modelling language. Thus ML verification is less costly both in human and computer time and promotes ease and reliability.

Modifiability:

This refers to determining a new modeller's form expressed in the modelling language, when there are changes in reality. The modelling language representation can be modified directly with ease. Thus linear program modification in modelling language system is less costly. The modelling language offers similar advantages with respect to modifiability as in the case of verifiability.

Documentability:

This refers to maintaining all relevant information pertaining to different forms and their relationships. Modelling language system avoids extra documentation. Thus the documentation task is made simple and easy.

Independence:

This refers to conceptual dependencies on particular algorithmic form. Modelling languages depend less on choice of algorithmic form and therefore modeller need not be inconvenienced by the need to know about particular algorithmic form.

Simplicity:

This refers to the need of any intermediate form between modeller's form and algorithmic form. In contrast to matrix generators, modelling languages require no intermediate form. Modelling languages interact with people through few simple forms thereby enabling easier modelling for linear programming[68].

Further, a modelling language specifies its data symbolically in some natural and familiar pattern similar to modeller's form. It is not concerned with explicit data and enforces distinction between symbolic and explicit data which leads to more flexible and convenient management of the data for linear programming. The symbolic representation of data promotes reliability by serving as a check on the validity of

the explicit data. In the modelling language system, the ML translator can detect explicit data that are not in conformance with symbolic data and hence any data error has good chance of being caught and corrected before an erroneous algorithmic form is generated. Modelling languages being independent of algorithmic form may employ more flexible notation based on subscripted or indexed identifiers for naming LP components. Use of indexed identifiers makes linear program modelling easier. Modelling languages promote row-wise organization based on variable-and-constraint (row-wise) form leading to greater flexibility and power^[68]. The evident weakness of modelling languages is that they are hard to implement. However, the benefits out-lined above represent the significant advantages of using a modelling language. In the light of the above advantages several prototype modelling languages have been under development, for example, ALPS [170], GAMS [105], LMC [122], LPMODEL [103], MAGIC [52], MGG [147], and UIMP [167] in the recent years. LAMP has been developed at IIT Kanpur, both as a vehicle for teaching/research and to try out our ideas on the synthesis of flexible data and model access, which in our view constitute the central theme of the decision support system.

4.2 DESIGN GOALS

A modelling language must fulfil the conflicting requirements imposed by the demands of the modeller on the one hand and the requirements imposed by the computers on the other. From the modeller's view the modelling language should be easy to use and understand. It should have terminology that is convenient to the modeller. The expressions of a modelling language need to be powerful and yet flexible. From the view point of the computer system, the modelling language should be capable of being processed and translated at reasonable cost and time. The syntax and semantics of the modelling language must be unambiguous and simple. The notation and symbols must be representable with the standard "ASCII" character set. Representative keywords would replace special symbols that have natural meaning to the environment such as "SIGMA" for summation. In order to make the language useful to several model builders, it must have a grammar which is simple and yet powerful. As the grammar specifies what constructs are acceptable within the language, a simple grammar will permit any potential user to learn the language with minimal effort in a relatively short period of time. Additionally, the grammar should be free from exceptions and still be general enough to handle the diverse needs of model builders or potential users.

Any practical modelling language is a compromise between the conflicting requirements imposed by modeller and computer system. This compromise can be carried out in many possible ways leading to various kinds of modelling languages, with greater priority for some specific features. The requirement of user convenience can be met ideally by using an existing modeller's form as a modelling language. Through several years of use and continuous refinement, common modeller's form have been made convenient and understandable and their notation offers the desired power and flexibility. However, these existing modeller's forms are intended entirely for human communication and are neither readable nor translatable by foreseeable computer systems. Like any natural language, they are also ambiguous, complex and their meanings are context dependent. Moreover, some important notations, such as, summation sign and subscripts are even now incompatible with ordinary computer hardware.

Nevertheless, an existing modeller's form is a reasonable point to start with for the design of an attractive and workable modelling language. Actually, such a form need only to be modified, so that it can be read and processed by the computer. Essentially, the variety and complexity of the existing modeller's form must be reduced, so that every expression has an unambiguous syntax and meaning^[68]. The choice of an underlying structure of the modeller's form is an

important decision in the design of a modelling language. Systems like MPOS [46] and LINDO [144], use "free-form" input, where the user specifies each of the constraint as if he were writing on a piece of paper. Such a structure while undoubtedly attractive, is very inefficient for large problems where recognizable structure occurs, which can be generated automatically. Traditionally in analytical work, one views linear programming as a set of constraints generally referred to as "row-wise form." Because of its flexibility, power and naturality, row-wise modeller's form is used as the underlying structure of representation in LAMP.

With these observations we can finally state our goals as:

- (i) The modelling language must be declarative i.e., the user should only define and describe the problem and no algorithm need be presented by him for formulation as that will be done by ML system in a standard way for all the models.
- (ii) The modelling language must be symbolic i.e., most of the problem elements - variables, constraints and objective, be representable using mnemonic symbols.
- (iii) The modelling language should be general, i.e., it should be capable of defining most of the linear programming models having a linear objective function to be maximized or minimized along with a set of linear constraints. Moreover,

it should be extendable to integer and nonlinear programming situations as well.

(iv) The modelling language should be understandable i.e., the syntax and semantic features should represent the language in such a way that the user will be able to comprehend it easily.

(v) The modelling language should be natural i.e., model can be expressed in terms natural to the domain of the problem.

(vi) The modelling language system must be readable on any interactive terminal with a limited standard key board character set.

(vii) The modelling language system must be sufficiently user friendly. A modern modelling language system should facilitate interactive session as well as batch operation. During the interactive session a modeller should be able to operate the system by typing commands at computer terminal; and the system should be designed to send relevant messages and results back to the terminal as it runs. A user friendly system should be designed to have message display pertaining to specific information from user, error messages flashed resulting from improper-user input and should have necessary recovery mechanism. In batch operation the user or modeller can submit the whole set of commands and model information using external files and get result when the entire job is completed.

Interactive session affords closer control of a system and gives a better feel for what is going on. A modeller can watch the model construction by having it report periodically or at designed stages on its progress. Interactive session is quite useful for model construction phase as the mistakes can be corrected as they turn up. Interactive systems consequently encourage experimentation which is essential to successful modelling. Ease of experimentation also makes an interactive system easier to learn. The modelling language system should concentrate on the construction and development of the model rather than on its solution. The modelling language system should have sufficient syntactic and semantic error analysis capabilities in a way to help the modeller find mis specification in his model.

(viii) The modelling language system should be based on modularity concept i.e., dividing the larger task into smaller sub tasks that can be treated separately. The modeller should not be restricted by any order in which the various components of a model need to be written, apart from the contextual naturalness considerations.

(ix) The modelling language should have abstraction capability i.e., the system should be capable of expressing nature of the problem independent of particular details such as numerical constant.

- (x) The modelling language system should have provision for models to be nested, so that the output of one model may be used as input to other model, if required.
- (xi) Any modification to existing modelling language system should be simple to implement.
- (xii) The modelling language system should be able to interface with existing databases in such a way that not more than a minimum of knowledge is required from the modeller.
- (xiii) The modelling language system should be portable, so that the model representation is independent of computing equipment.
- (xiv) The modelling language system must meet atleast the following features with reference to linear programs.
 - (a) Symbolic identification of linear program components i.e., constraints, variables, R.H.S. and objective.
The abstract model form of constraint or objective is to be modelled in row-wise manner.
 - (b) Grouping of variables and constraints i.e., the system has to permit variable grouping and a particular type of constraint grouping.
 - (c) Indexing expression i.e., specifying how a group of expression is indexed.

- (d) Ability to parametrically specify constraints, variables i.e., the system has to build abstract constraints with parameters such as "ALPHA", indexed coefficient and variable.
- (e) Special features: e.g., system should have provision for simple and generalized upper bounds, time lag/lead features. Most of the linear programs are likely to have simple upper bounds and generalized upper bounds. Also, some linear programming situation may necessitate the incorporation of time lag/lead aspect to model a wide variety of real life applications.
- (f) Interface with standard problem solving software for linear program and a standard database management software: The model generated by the modelling language system needs to be solved for optimization using standard problem solving software. Further the result so obtained as well as input data need to be utilized for generation of related desired information and answering query raised by the user. Effective query system necessitates the use of a standard database management system software.
- (g) Automatic generation of coefficients in special cases such as generalized and simple upper bounds: In order to minimize the coefficient values supplied by the user during interactive session or through external file in

batch operation, the system should generate appropriate coefficient values at the run time for some special cases of generalized or simple upper bounds.

- (h) Blending constraints: i.e., the system should have provision for blending type constraints commonly required for modelling in process industry environment where the output of one process becomes the input for the next process.

The above features have been adopted as the design goals in our implementation. They were arrived at after extensive study of the existing systems under development elsewhere. The major emphasis, however, is on proving the ideas and not on the development of a software product, which calls for much larger investment of a different skill.

4.3 IMPLEMENTATION:

4.3.1 Introduction:

The modelling language system, LAMP, has been implemented on DEC-10 system at IIT Kanpur. Most of the code is written in PASCAL [98]. The exercise has been primarily to prove the ideas, rather than developing a full fledged software product. Hence the diagnostic capabilities of the model are limited. However during interactive session or in batch operation error and other relevant messages are flashed back for modeller's or user's help. The system has, however been well tested.

The LAMP system uses LINDO [145] and GINO [114] as problem solving software for linear and nonlinear programs respectively. The data management software is the package UNIFY [168] that runs on the UNIX [169] based machine UPTRON S-32, available at IIT Kanpur which is networked to DEC-10 machine.

The actual model building is in two stages, model structure definition and model data definition. In the first stage, modeller defines the terminology and then the abstract structure of the model using a simple syntax detailed in Section 4.3.2. In the model data definition phase the data are generated by executing the model structure with the data on files or provided by modeller during interactive session to produce the actual data for interfacing to a problem solving software. The format of the generated data are an industry standard MPS file [92] and row-wise equation form for input as well as a relational format discussed in the next chapter for both input and solution values, which we believe would be the future generation standard. The salient features of implementation are described below.

4.3.2 The Grammar:

The syntax of LAMP has been kept close to the way the mathematical programs are typically written by most modellers. The summation symbol has been replaced by "SUM" to enable one to use LAMP even with ordinary terminals. The other parts of

the modelling language use natural symbols. Many of the ideas are based on LPMODEL [103]:

The basic building block is an atom, a group of atoms constitute a molecule. Molecules are typically used as index variables. A typical example would be the molecule "MONTH" consisting of atoms "MAY", "JUNE" and "JULY". The variables, constraints and objective function are built out of identifiers which may be scalar or arrays, indexed over the molecules. To provide maximum flexibility, LAMP allows different molecules to have common atoms and provides automatic generation of cartesian products for summation. LAMP also provides for variable to appear on right hand side when it is natural to modelling. Multi-level indexing of identifiers, multiple terms and parametric terms are also permitted by LAMP. The values corresponding to coefficient identifiers are stored in files (possibly obtained from corporate data base of the organization through the use of some database management software) or provided by user during interactive session and the matrix generated at the time of execution of LAMP. LAMP provides for automatic generation of coefficients in the case of specially structured constraints such as simple upper bounds or generalized upper bounds.

A compact and complete way to describe the modelling language used in LAMP is accomplished by providing

its grammar. The grammar is a set of rules that determines which are the valid constructs in the language. It describes the syntax. Moreover, one still needs to explain the meaning of the constructs, usually referred to as semantics. For user's familiarity with the grammar, the EBNF (Extended Backus Naur Form) syntax for terminology, objective and constraint — equations and assignments has been provided. Hopefully it is to be used as the key to a quick understanding of what constructs are syntactically correct. The detailed syntax of the language in Standard EBNF form [98] is given in Figure 1.

The modelling language grammar is written using standard notation (pattern) of EBNF and contains the following symbols known as meta-symbols.

<u>Meta-Symbol</u>	<u>English Equivalent</u>	<u>Use</u>
=	is defined to be	Separates a phrase name from its definition
	alternatively (or)	Separates alternative definition of a phrase
.	period or full stop	end of production

Additionally, the meta-symbol [...] refers to the optional (zero or one) occurrence of enclosed construct and {...}

refers to zero or more occurrences of enclosed construct.

The meta-symbol (X|Y) refers to a grouping: either X or Y.

The meta-symbol "XYZ" refers to the terminal symbol XYZ.

* !! NOTE : Lines starting with '*' are 'Comment' lines.

*

Metasymbol	Meaning
=	is defined to be
!	alternatively
.	end of production
[X]	0 or 1 instance of X
{X}	0 or more instances of X
(X Y)	a groupings: either X or Y
XYZ	the terminal symbol XYZ

```

Problem      = "Problem."
              Terminology
              Objective
              Constraint_sequence .

Terminology  = Molecule_sequence
              Identifier_sequence .

Molecule_sequence = Molecule_definition"eoln"({Molecule_definition
              "eoln"}).

Molecule_definition = Molecule_name"="Atom_name{"Atom_name"}.
Molecule_name      = Name.
Atom_name           = Name.
Name                = Letter{Alphanumeric}.
Letter              = ("a"|"b"|"..."|"z").
Alphanumeric        = Letter|Digit.
Digit               = ("0"|"1"|"..."|"9").

Identifier_sequence = Identifier"$"eoln"({Identifier"$"eoln}).

Identifier     = Name{"."Molecule_name"}.

Objective      = Operator LHS"eoln".
Operator       = ("MIN"|"MAX").
LHS            = Term["+Term_sep"].
Term_sep       = (Term|Param_term).
Param_term     = Parameter Term .
Term           = "[Summation_id":Coefficient_id"*Variable_id
              ("?"|"??")(":"|"")".

Parameter      = Name.
Summation_id   = Molecule_name{"":Molecule_name}.
Coefficient_id  = Identifier.
Variable_id    = (Molecule_name|Identifier).

Constraint_sequence = General_sequence{Spl_sequence}.
General_sequence    = General_constraint"eoln"({General_constraint
              "eoln"}).
General_constraint   = "SUM"LHS(":"|"=")RHS.
RHS                 = Identifier.
Spl_sequence        = Spl_constraint"eoln"({Spl_constraint"eoln"}).
Spl_constraint       = {Bounds_constraint}{simple_sum_constraint}.
Bounds_constraint    = Bounds_constraint"eoln"({Bounds_constraint"eoln"}).
Bounds_constraint    = (Molecule_name|Identifier)"?"(":"|"=")RHS.
Simple_sum_constraint = Simple_sum_constraint"eoln"({Simple_sum_constraint
              "eoln"}).
Simple_sum_constraint = "SUM"[Summation_id":Variable_id"?"]
              (":"|"=")RHS.

```

*

Fig. 1: EBNF Grammar for LAMP.

In the Extended Backus Naur Form (EBNF) the meta-symbol $\langle \dots \rangle$ enclosing non-terminal symbols have been dropped and any symbol which does not appear on the left side of a production is a terminal symbol. The symbol "eoln" is used to indicate the end-of-line feature.

Every effort has been taken to keep the representation as natural as possible i.e., the model can be described in terms natural to the domain of the problem. The language is based on limited character set usually available on standard terminals e.g., (no Greek letters). It permits no subscripts and superscripts. It allows algebraic operators "+" for addition and "*" for multiplication. Using above described meta-symbols as primitives and constructing multi-character names for variables, identifiers and parameters, the modelling language builds the model as algebraic equation or expression which represents the real life problem under consideration. For example, a typical constraint set could be

$$\text{SUM}[\text{MONTH}; \text{FERTREQ.CROP.MONTH} * \text{MONTH}^2] \leq \text{TFERTREQ.CROP}$$

indicating that the fertilizer required (FERTREQ) summed over all the months must be limited to the total fertilizer (TFERTREQ) required for every crop. The coefficients indicating the fertilizer requirement come from the indexed identifier (FERTREQ.CROP.MONTH). The molecule name to the left of the symbol ":" i.e., "MONTH," represents the summation set.

The decision variables are indicated by "MONTH?". The actual decision variable corresponding to this constraint will be all the atoms associated with the molecule name "MONTH". It may be noted that the model structure is independent of data i.e., the structure of the constraint set would be independent of the number of months considered for planning.

4.3.3 Salient Features:

The main features of the LAMP system are as follows:

(i) Indexed Constraints:

LAMP allows a simple definition of constraints indexed by one or more molecules, permitting multi-level indexing (upto three levels are permitted in the current implementation).

Example:

$$\text{SUM}[\text{CROP}; \text{LABORREQ.CROP.MONTH} * \text{CROP ?}] \leq \text{LABORAVL.MONTH}$$

This abstract form of the constraint will finally generate as many no. of constraint inequalities as the number of atoms for molecule 'MONTH', labelling each one of them by particular atom names corresponding to molecule "MONTH".

(ii) Sum of Terms:

LAMP allows a constraint or a set of constraints to be composed of more than one symbolic term.

Example:

$$\begin{aligned} & \text{SUM}[\text{CROP}:\text{WATERREQ}.\text{CROP}.\text{MONTH}*\text{CROP} ?] + \\ & [\text{LIVSTOCK}:\text{WATERREQ}.\text{LIVSTOCK}.\text{MONTH}*\text{LIVSTOCK} ?] \\ & \leq \text{TOTWATER}.\text{MONTH} \end{aligned}$$

This abstract form of the constraint will finally expand both the terms in row-wise manner, firstly "CROP" atoms as variables and later "LIVSTOCK" atoms as variables indicating that the water required summed over all crop atoms and water required summed over all livestock atoms must be limited to the total water available (TOTWATER.MONTH) for every month. The number of constraints generated will be same as the number of atoms for molecule "MONTH".

(iii) Sum of Terms with Constant Multipliers:

For parametric analysis one would like to have provision for a constant multiplier to be included in the model. LAMP is capable of handling such feature.

Example:

$$\begin{aligned} & \text{MAX}[\text{CROP}:\text{PROFIT}.\text{CROP}*\text{CROP} ?] + \\ & \text{ALPHA}[\text{LIVSTOCK}:\text{PROFIT}.\text{LIVSTOCK}*\text{LIVSTOCK} ?] \end{aligned}$$

In this abstract form of the constraint "ALPHA" has been used as a parameter. The value for this will be provided by user at run-time.

(iv) Blending Constraint:

In process industries blending type constraints are frequently used to represent process energy or resource balance.

Example:

$$\begin{aligned} \text{SUM}[\text{PROCESS1:HEAT}.\text{PROCESS1*PROCESS1?}] + \\ \text{ALPHA}[\text{PROCESS2:HEAT}.\text{PROCESS2*PROCESS2?}] \\ = \text{ZERO} \end{aligned}$$

(v) Bounds:

Most mathematical programs contain bounds. Special algorithms like bounded variable algorithms exploit this structure. LAMP permits for bounds as a special constraint and the modeller need not provide the coefficients which are automatically generated at run-time.

Example:

$$\text{CROP?} \leq \text{CEILING.CROP}$$

This abstract form of the constraint will generate a set of simple upper bound constraints. The number of constraints generated will be same as the number of atoms for "CROP" molecule.

(vi) Simple Summation:

A good number of constraints in any mathematical program is in the form of a simple summation of a set of variables. In order to save the unnecessary entry of these coefficients

in the data file, LAMP permits automatic generation of such unit coefficients.

Example:

```
SUM[CROP:CROP?] <= TOTLAND
```

This abstract form of the constraint will generate a single constraint with all coefficients as unity for each atom of the "CROP" molecule and the summation must be limited to land available for "CROP" i.e., (TOTLAND).

(vii) Special Features:

Special features include the generation of combinatorial constraints. The cartesian product of indices can be indicated in the model structure rather than explicitly defining a molecule listing of the elements of the cartesian product.

Example:

```
MIN[ROW: COLUMN: COST. ROW. COLUMN * ROW. COLUMN ?]
```

This abstract form of the objective function will generate the cartesian product corresponding to ROW. COLUMN and using double summation concept write the objective function.

(viii) Restricted Summation:

Restricted summation can be handled by defining another molecule consisting of restricted atoms.

Example:

MIX = APPLE, MANGO

This indicates that molecule "MIX" is defined to have atoms as "APPLE" and "MANGO". Then the abstract form of the constraint

SUM[MIX:MIX ?] <= CEILMIX

will generate a single constraint of simple summation type i.e., similar to generalized upper bound constraint comprising of only two variables viz. "APPLE" and "MANGO".

(ix) Nonlinear Terms:

Nonlinear terms can be handled by interactively specifying the form of nonlinearity of objective function/constraints. The current implementation allows simple polynomials and standard trigonometric functions only. The variables associated with nonlinearity is indicated by double question mark (??) instead of single question mark (?).

Example:

MAX[CROP:WATER.CROP*CROP??]

This abstract form of the objective function will generate the objective function expression after getting the nonlinear parameter information from the user at run-time.

(x) Variable Right Hand Side:

Variable right hand side is allowed by LAMP which may sometimes be required for meaningful constraint modelling. Variable on the right hand side feature is indicated by brace (}) closing instead of bracket (]) closing used in the abstract form of the constraint.

Example:

SUM[CROP:WATER.CROP*CROP ?} <= TOTWATER

This abstract form will generate constraint expression having variable on the right hand side after getting the right hand side variable name information from the user at run-time.

4.4 ILLUSTRATIVE EXAMPLE

4.4.1 Sample Problem:

To illustrate the capability of the LAMP system, we will use the following agricultural planning example adopted from [103] and modified slightly to represent LAMP capabilities.

A farm manager is planning for the next season consisting of the periods May, June and July. The various crops for the season under consideration being cotton and onion (cultivation crops grown in a field) and apples and oranges (grown in an orchard). The land availabilities for the field and the orchard are 1850 and 600 units out of the total 2700 units, the balance being used for grazing. The labour availability

is limited to 5850 units. The total availability of water is limited to 205000, 265000 and 275000 units in the months of May, June and July respectively, out of which 200000, 260000 and 270000 units only can be used for crops. The corresponding availability figures for fertilizer are 4000, 5000 and 6000 units. The farm is maintaining livestock i.e., cattle and sheep also. The per unit profit contributions per unit area of cultivation and similar figures for the livestock type are known. There are bounds on the production level of the crops and livestock. In addition, a minimum quantity of production must be ensured. The per unit requirement of water, labour and fertilizer are given. The entire data is summarized in Table 1 to Table 7. The farm manager is interested in an optimal strategy that maximizes his revenue.

Table 1: Land Availability

Field	Orchard	Total
1850	600	2700

Table 2A: Average Unit Labour Requirement

Cotton	Onion	Apple	Orange
2.9	2.7	1.0	1.5

Table 2B: Labour Availability

Total
5850

Table 3A: Unit Water Requirement.

	Cotton	Onion	Apple	Orange	Cattle	Sheep
May	65	-	-	-	0.1	0.4
June	80	60	50	75	0.2	0.5
July	90	-	64	85	0.3	0.6

Table 3B: Water Available for Crops

	Quantity
May	200000
June	260000
July	270000

Table 3C: Total Water Available

	Quantity
May	205000
June	265000
July	275000

Table 4A: Unit Fertilizer Requirements.

	Cotton	Onion	Apple	Orange
May	1	4	7	10
June	2	5	8	11
July	3	6	9	12

Table 4B: Fertilizer Availability

	Quantity
May	4000
June	5000
July	6000

Table 5: Unit Profit Contribution.

Cotton	Onion	Apple	Orange	Cattle	Sheep
6453	6110	4814	8812	500	600

Table 6A: Bounds for Crops

Cotton	Onion	Apple	Orange
2000	250	500	800

Table 6B: Bounds for Livestock

Cattle	Sheep
400	300

Table 7A: Yield Data

Cotton	Onion	Apple	Orange
12	13	14	15

Table 7B: Minimum Production

Quantity
15000

4.4.2 LAMP Formulation of the Sample Problem:

The sample problem discussed in sub section 4.4.1 is written in LAMP using grammar syntax specified in Figure 1. The terminology phase appears in Figure 2 and the abstract model structure phase comprising of objective and constraints appear in Figure 3.

The various phases of model formulation for the above mentioned agricultural planning sample problem is further explained in detail. Based on specific requirement(s) of the sample problem, Fig. 1 representing terminology phase depicts the description of various molecules and identifiers. The molecules defined are "MONTH" with "MAY", "JUNE" and "JULY" as its atoms; "CROP" with "COTTON", "ONION", "APPLE" and "ORANGE" as its atoms; "FIELD" with "COTTON" and "ONION" as its atoms; "ORCHARD" with "APPLE" and "ORANGE" as its

```

=====
*      TERMINOLOGY PHASE
*      MOLECULE
=====

MONTH=MAY,JUNE,JULY

CROP=COTTON,ONION,APPLE,ORANGE

FIELD=COTTON,ONION

ORCHARD=APPLE,ORANGE

LIVESTOCK=CATTLE,SHEEP

RESOURCE=LAND,LABOR,WATER,FERTILZER

=====
*      IDENTIFIER
=====

*      LAND AVAILABILITY :                ( Table-1 )

TOTALAVL$

FIELDVL$

ORCHAVL$

*      LABOR :                            ( Table-2 )

LABORREQ.CROP$                            ( Table-2A )

LABORAVL$                                ( Table-2B )

*      WATER :                            ( Table-3 )

WATERREQ.CROP.MONTH$                     ( Table-3A )

WATERFER.LIVESTOCK.MONTH$                ( Table-3A )

TOTWATER.MONTH$                          ( Table-3B )

WATERAVL.MONTH$                          ( Table-3C )

*      FERTILZER :                        ( Table-4 )

FERTREQ.CROP.MONTH$                      ( Table-4A )

FERTAVL.MONTH$                           ( Table-4B )

*      UNIT PROFIT :                      ( Table-5 )

PROFIT.CROP$

PROFIT.LIVESTOCK$

*      BOUNDS :                           ( Table-6 )

CEIL.CROP$                               ( Table-6A )

CEIL.LIVESTOCK$                           ( Table-6B )

*      YIELD :                            ( Table-7 )

YIELD.CROP$                              ( Table-7A )

MINPROD$                                 ( Table-7B )

```

Fig. 2: Agricultural Problem Terminology Phase.

```

*=====
*      PROBLEM STRUCTURE
*=====
*      OBJECTIVE
*=====
MAX [CROP:PROFIT.CROP*CROP?] + ALPHA[LIVSTOCK:PROFIT.LIVSTOCK*LIVSTOCK?]
*=====
*      CONSTRAINTS
*=====
*      Wateravailability(Total)
SUM[CROP:WATERREQ.CROP.MONTH*CROP?] + [LIVSTOCK:WATERREQ.LIVSTOCK.MONTH*LIVSTOCK?]
?]<=TOTWATER.MONTH
*      Labouravailability
SUM[CROP:LABORREQ.CROP*CROP?] <= LABORAVL
*      Landavailability
SUM[CROP:CROP?] <= TOTALAVL
*      Fieldlandavailability
SUM[FIELD:FIELD?] = FIELDAVL
*      Orchardlandavailability
SUM[ORCHARD:ORCHARD?] = ORCHAVL
*      Wateravailability for crops
SUM[CROP:WATERREQ.CROP.MONTH*CROP?] = WATERAVL.MONTH
*      Minimum yields
SUM[CROP:YIELD.CROP*CROP?] >= MINPROD
*      Fertilizer consumption
SUM[CROP:FERTREQ.CROP.MONTH*CROP?] = FERTAVL.MONTH
*      Ceilings on crops
CROP? <= CEIL.CROP
*      Ceilings on livestock
LIVSTOCK? <= CEIL.LIVSTOCK
*=====

```

Fig. 3: Agricultural Problem Model Structure.

atoms "LIVSTOCK" with "CATTLE" and "SHEEP" as its atoms "RESOURCE" with "LAND", "LABOR", "WATER" and "FERTILIZER" as its atoms. Then the identifiers defined are "TOTALAVL", "FIELDAVL" and "ORCHAVL" to represent land availability features based on data from Table 1; "LABORREQ.CROP" and "LABORAVL" to represent labour features based on data from Table 2A and Table 2B; "WATERREQ.CROP.MONTH", "WATERREQ.LIVSTOCK.MONTH", "TOTWATER.MONTH" and "WATERAVL.MONTH" to represent features related with water resource based on data from Table 3A, Table 3B and Table 3C; "FERTREQ.CROP.MONTH", and "FERTAVL.MONTH" to represent fertilizer features based on data from Table 4A and Table 4B; "PROFIT.CROP" and "PROFIT.LIVSTOCK" to represent unit profit features based on Table 5; "CEIL.CROP" and "CEIL.LIVSTOCK" to represent bounds information based on data from Table 6A and Table 6B; "YIELD.CROP" and "MINPROD" to represent yield features based on data from Table 7A and Table 7B.

Next in the abstract model structure phase first the objective function equation is developed and then the whole bunch of constraints representing the real world problem as a linear program is developed. However, the constraints need not be in any particular order i.e., the modeller can develop the constraints in abstract form in any sequence. In order to fulfil the farm manager's strategy of maximizing his revenue in the example problem, the LAMP representation of objective function will be as,

$$\text{MAX}[\text{CROP:PROFIT.CROP} \times \text{CROP}] + \\ \text{ALPHA}[\text{LIVSTOCK:PROFIT.LIVSTOCK} \times \text{LIVSTOCK}]$$

The above abstract form representation of the objective function will generate a single expression considering profit contribution coming from crops and livestock. The actual data values for coefficients will be coming from the associated identifier data values stored in the system during run-time. The value of the parameter, "ALPHA" will be supplied by user as 1.0 or any desired value at run-time. Based on above information, LAMP system generates the exact expression for objective function.

Now considering the total water availability as resource, the LAMP representation of associated restriction in abstract form will be as

$$\text{SUM}[\text{CROP:WATERREQ.CROP.MONTH} \times \text{CROP}] + \\ + [\text{LIVSTOCK:WATERREQ.LIVSTOCK.MONTH} \times \text{LIVSTOCK}] \\ \leq \text{TOTWATER.MONTH}$$

The above abstract form of the constraint will generate three inequalities, one corresponding to each month (i.e., "MAY", "JUNE", "JULY"). The actual coefficient values will be coming from the associated identifier data values stored in the system during run-time. Similarly, considering the labour availability as resource for crops, the LAMP representation of associated restriction will be as

$$\text{SUM}[\text{CROP:LABORREQ.CROP} \times \text{CROP}] \leq \text{LABORAVL}$$

The above abstract form representation of constraint will generate an inequality with "CROP" atoms (e.g., "COTTON", "ONION", "APPLE", "ORANGE") as variables. The actual coefficient values will be coming from the associated identifier data values stored in the system during run-time.

Now considering the land availability as resource i.e., total land, field land and orchard land as separate resource, the LAMP representations of the associated restriction will be as

$$\begin{aligned} \text{SUM}[\text{CROP:CROP ?}] &\leq \text{TOTALAVL} \\ \text{SUM}[\text{FIELD:FIELD ?}] &\leq \text{FIELD AVL} \\ \text{SUM}[\text{ORCHARD:ORCHARD ?}] &\leq \text{ORCHAVL} \end{aligned}$$

Each of the above abstract form will generate single inequality having unity (i.e., 1.0) as all coefficient values, generated by the system at run-time.

Now considering the water availability as resource only for crops, the LAMP representation of associated restriction in abstract form will be as

$$\begin{aligned} \text{SUM}[\text{CROP:WATERREQ.CROP.MONTH*CROP ?}] \\ &\leq \text{WATERAVL.MONTH} \end{aligned}$$

The above abstract form of the constraint will generate three inequalities, one corresponding to each month (i.e., "MAY", "JUNE", "JULY") with "CROP" atoms (i.e., "COTTON", "ONION", "APPLE", "ORANGE") as variables. The actual coefficient values

will be coming from the associated identifier data values stored in the system during run-time.

Now considering the minimum yield requirement, the LAMP representation of associated restriction in abstract form will be as

$$\text{SUM}[\text{CROP}; \text{YIELD.CROP} * \text{CROP} ?] \geq \text{MINPROD}$$

The above abstract form of the constraint will generate an inequality with "CROP" atoms (i.e., "COTTON", "ONION", "APPLE", "ORANGE") as variables. The actual coefficient values will be coming from the identifier data values stored in the system during run-time.

Now considering the fertilizer as resource, the LAMP representation of associated restriction in abstract form will be as

$$\begin{aligned} \text{SUM}[\text{CROP}; \text{FERTREQ.CROP.MONTH} * \text{CROP} ?] \\ \leq \text{FERTAVL.MONTH} \end{aligned}$$

The above abstract form will generate three inequalities, one corresponding to each month (i.e., "MAY", "JUNE", "JULY") with "CROP" atoms (i.e., "COTTON", "ONION", "APPLE", "ORANGE") as variables. The actual coefficient values will be coming from the associated identifier data values stored in the system during run-time.

Finally, considering the various bounds e.g., ceiling on crop and ceiling on livestock, the LAMP representation of

associated restrictions in abstract form will be as

$$\text{CROP?} < = \text{CEIL.CROP}$$

$$\text{LIVSTOCK?} < = \text{CEIL.LIVSTOCK}$$

Based on the terminology phase definition of molecule "CROP" having four atoms (i.e., "COTTON", "ONION", "APPLE", "ORANGE") and molecule "LIVSTOCK" having two atoms (i.e., "CATTLE", "SHEEP") the above abstract forms will generate four and two inequalities respectively. Similar to simple upper bound constraint, each of the four inequalities will be having single variable as atom of "CROP" molecule and each of the other two inequalities will be having single variable as atom of "LIVSTOCK" molecule.

Thus the entire formulation for the example problem as a linear program is completed. The LAMP system generates the formulation both in MPS format as well as in expanded inequality form having entries only corresponding to nonzero coefficient values. These appear as Figure 4 and Figure 5 respectively.

4.5 FURTHER EXAMPLES:

To demonstrate the full capabilities of LAMP to model a variety of real life situations, three sample problems were selected from Ozon [131]. These are "Foundry Charging Problem" representing simple product mix decision; "Gasoline Blending Problem" that uses blending concept in a process

* !! NOTE : Lines starting with '*' are 'Comment' lines.

```

*      MODEL FORMULATION IN MPS FORMAT for Agricultural Model
*
NAME    LINE GENERATED  MPS FILE( MAX)
ROWS
N 1
L 2
L 3
L 4
L 5
L 6
L 7
L 8
L 9
L 10
L 11
G 12
L 13
L 14
L 15
L 16
L 17
L 18
L 19
L 20
L 21
COLUMNS
COTTON      1      6453.00006
COTTON      2      65.00000
COTTON      3      80.00000
COTTON      4      90.00000
COTTON      5      2.90000
COTTON      6      1.00000
COTTON      7      1.00000
COTTON      9      65.00000
COTTON     10      80.00000
COTTON     11      90.00000
COTTON     12      12.00000
COTTON     13      1.00000
COTTON     14      2.00000
COTTON     15      3.00000
COTTON     16      1.00000
ONION       1      6110.00001
ONION       3      60.00000
ONION       5      2.69999
ONION       6      1.00000
ONION       7      1.00000
ONION      10      60.00000
ONION      12      12.99999
ONION      13      4.00000
ONION      14      5.00000
ONION      15      6.00000
ONION      17      1.00000
APPLE       1      4814.00001
APPLE       3      50.00000
APPLE       4      63.99999
APPLE       5      1.00000
APPLE       6      1.00000
APPLE       8      1.00000
APPLE      10      50.00000
APPLE      11      63.99999
APPLE      12      14.00000
APPLE      13      7.00000
APPLE      14      8.00000
APPLE      15      9.00000
APPLE      18      1.00000

```

Fig. 4: Model Formulation in MPS Format for Agricultural Problem (Contd.).

```

*  !! NOTE : Lines starting with '*' are 'Comment' lines.
*  MODEL FORMULATION IN MPS FORMAT for Agricultural Model ...(continued)
*  -----
ORANGE      1      8812.99996
ORANGE      3      75.00000
ORANGE      4      85.00000
ORANGE      5       1.50000
ORANGE      6       1.00000
ORANGE      8       1.00000
ORANGE     10      75.00000
ORANGE     11      85.00000
ORANGE     12      15.00000
ORANGE     13      10.00000
ORANGE     14      10.99999
ORANGE     15      12.00000
ORANGE     19       1.00000
CATTLE      1      600.00000
CATTLE      2       0.10000
CATTLE      3       0.20000
CATTLE      4       0.30000
CATTLE     20       1.00000
SHEEP       1      719.99999
SHEEP       2       0.40000
SHEEP       3       0.50000
SHEEP       4       0.60000
SHEEP      21       1.00000
RHS
RHS         2      205000.00119
RHS         3      265000.00059
RHS         4      275000.00000
RHS         5      5850.00000
RHS         6      2699.99998
RHS         7      1849.99999
RHS         8       600.00000
RHS         9      200000.00000
RHS        10      250000.00000
RHS        11      269999.99981
RHS        12      15000.00000
RHS        13      4000.00000
RHS        14      5000.00000
RHS        15      6000.00000
RHS        16      2000.00000
RHS        17      250.00000
RHS        18      500.00000
RHS        19      800.00000
RHS        20      400.00000
RHS        21      300.00000
ENDATA
*  -----

```

Fig. 4: Model Formulation in MPS Format,
for Agricultural Problem.

```

* 11 NOTE : Lines starting with '*' are 'Comment' lines.
* -----
MAX 6453.00006COTTON + 6110.00001ONION + 4814.00001APPLE + 8812.99996ORANGE
+ 600.00000CATTLE + 719.99999SHEEP

ST.
65.00000COTTON + 0.10000CATTLE + 0.40000SHEEP <=205000.00119
80.00000COTTON + 60.00000ONION + 50.00000APPLE + 75.00000ORANGE
+ 0.20000CATTLE + 0.50000SHEEP
=265000.00059
90.00000COTTON + 63.99999APPLE + 85.00000ORANGE + 0.30000CATTLE
+ 0.60000SHEEP =275000.00000
2.90000COTTON + 2.69999ONION + 1.00000APPLE + 1.50000ORANGE
= 5850.00002
1.00000COTTON + 1.00000ONION + 1.00000APPLE + 1.00000ORANGE
= 2699.99998
1.00000COTTON + 1.00000ONION = 1849.99999
1.00000APPLE + 1.00000ORANGE = 600.00000
65.00000COTTON =200000.00000
80.00000COTTON + 60.00000ONION + 50.00000APPLE + 75.00000ORANGE
=259999.99940
90.00000COTTON + 63.99999APPLE + 85.00000ORANGE =269999.99880
12.00000COTTON + 12.99999ONION + 14.00000APPLE + 15.00000ORANGE
= 15000.00000
1.00000COTTON + 4.00000ONION + 7.00000APPLE + 10.00000ORANGE
= 4000.00000
2.00000COTTON + 5.00000ONION + 8.00000APPLE + 10.99999ORANGE
= 5000.00000
3.00000COTTON + 6.00000ONION + 9.00000APPLE + 12.00000ORANGE
= 6000.00000
1.00000COTTON = 2000.00000
1.00000ONION = 250.00000
1.00000APPLE = 500.00000
1.00000ORANGE = 800.00000
1.00000CATTLE = 400.00000
1.00000SHEEP = 300.00000

***** LIST OF VARIABLES
COTTON
ONION
APPLE
ORANGE
CATTLE
SHEEP
* ***** LIST OF CONSTRAINT NAMES
TOTWATERMAY
TOTWATERJUNE
TOTWATERJULY
LABORAVL
TOTLAND
FILLAND
ORCHLAND
WATERAVLMAY
WATERAVLJUNE
WATERAVLJULY
REQYIELD
FERTAVL MAY
FERTAVL JUNE
FERTAVL JULY
CEIL COTTON
CEIL ONION
CEIL APPLE
CEIL ORANGE
CEIL CATTLE
CEIL SHEEP
* ?????????????????
* -----

```

Fig. 5: Model Formulation in Expanded Form for Agricultural Problem.

industry; "Bakery Problem" representing product planning with multiple stages. These problems alongwith their LAMP formulation are given in Appendix A.

A "Help" file to familiarize a novice user is provided in Appendix B which contains an interactive session using LAMP.

The detailed documentation listing the function of the various modules that constitute the LAMP system forms Appendix C.

CHAPTER V

STANDARD INTERFACE FOR DATA MANAGEMENT

5.1 INTRODUCTION:

Traditionally the problem processors were rigid computer programs designed for the numerical solution only. Later they evolved into matrix generators and then modelling languages. Many of the matrix generators were built around specific mathematical programming systems. It is generally true of modelling languages also. In order that the modelling languages/matrix generators can interface with problem processor, no standards are yet available. Traditionally in this area the input data representation employed by IBM's Mathematical Programming Systems Extended [92] has become a "de facto" standard. Unfortunately this standard is ancient and does not easily lend itself to further machine processing. We propose an alternate formalism using a simple relational view of the data. This view while acting as a standard also helps us significantly in the use of data management capabilities for other activities as well. Before we discuss the actual representations we take stock of some of the concepts of data base in general and relational systems in particular.

5.2 DATABASE MANAGEMENT SYSTEMS:

Data Base Management Systems (DBMS) usually are not thought as modelling systems by the OR/MS community, but they are all based on one or another data model [164] that attempts to organize virtually all of the data needs of various models. The actual model employed, viz. hierarchical, network or relational is important in the design of the user interface. The additional requirements of quality, integrity, security and control of data, have far reaching effects on the overall cost, accessibility and performance of any data base management system.

The data base management has the capability:

- . to make an integrated collection of data available to a wide variety of users;
- . to provide for quality and integrity of the data;
- . to ensure retention of privacy through security measures within the system;
- . to allow centralized control of the data base, which is essential for efficient data administration;
- . to provide data independence.

Database management systems have the attractive feature of introducing an independence between data structure and data manipulation. In the model management concept we

would like to have the structure of the model and the data independently kept and linked only at the time of actually running the model. This provides us an ability to retrieve and update data with an elegance and power that surpass qualitatively any matrix generator type software currently available. The support which a modeller gets in such tasks as data validation and manipulation is truly remarkable, with the availability of DBMS Software.

In database technology, the data definition generally consists of statement of the names of elements, their properties (e.g., string or numeric type), their relationship to other elements which make up the database. The data definition of a specific database is often called as schema. The schema provides the logical and physical structure. The logical structure describes the user's view of data. It deals with the named elements and their relationships rather than with the physical implementation. The physical structure describes the way the data values are stored within the system. It deals with pointers, character representation, floating point and integer representation, signed/unsigned representation, record blocking and access methods.

Logical data independence represents the ability to make logical change to the database without significantly affecting the programs which access it. Physical data

independence relieves user, from providing information on details of the structure and allows dealing with different structures and/or different access methods.

The logical structure which describes the user's view is based on data model concepts which allows user to define relationship existing among different classes of objects. Prominent data models are hierarchical, network, and relational as discussed earlier. The general trend in database management is that users are becoming increasingly oriented toward information content of their data and less concerned with its representational details. Such trend away from representation details is known as data independence. Database management systems using hierarchical or network structure represent the information by having contents of records; the connections between records and the ordering of records. User queries made to the DBMS are then framed in terms which depend on user knowledge of chosen representation. In the relational model, however, information is represented by data values only at the user interface. The user queries become free of any dependence on internal representation and may be framed in a high-level non-procedural language. At the same time, the system becomes free to choose any physical structure for storage of data and to optimize the execution of a given query. The various types of data models have their own advantages and disadvantages. It appears that the

relational model is likely to be the most widely used model in the future. It also has an attraction of an elegant structure and a strong theoretical foundation. Hence we limit ourselves to relational model in our work.

5.3 RELATIONAL DATABASE MANAGEMENT SYSTEMS:

E.F. Codd [43] first suggested relations (tables) as a tool for database management in 1970 and defined rigorously n -ary relations (i.e., tables having " n " columns representing " n " attributes) in the database context. He emphasized their advantages for data independence and symmetry of access. Codd's work introduced concepts which set the direction for research in relational database management in later years. The paper [43] defined a data sublanguage as a set of facilities, suitable for embedding in a host programming language which permits the retrieval of various subsets of data from the database. He recommended the first order predicate calculus, a logical notation to be an appropriate data sub language for n -ary relations. The paper also introduced a set of operators ("join", "projection", "selection", etc.) which were later developed into the well-known relational algebra. Finally, the paper explored the properties of "redundancy" and "consistency" of relations which laid the ground work for Codd's later theory of "normalization". The basic concepts and definitions underlying the relational data model are summarized below.

Mathematically, the term relation may be defined as: Given sets D_1, D_2, \dots, D_n (not necessarily distinct), a relation R is a subset of n -tuples each of which has its first element from D_1 , second element from D_2 , etc. The sets D_i are called domains. The number " n " is called the degree of R , and the number of tuples in R is called its cardinality.

It is customary to represent a relation as table in which each row represents a tuple. In the tabular representation of a relation the properties to be maintained are: no two rows are identical; the ordering of rows is not significant and the ordering of columns is significant. In a "relation" represented as a table, the number of rows is its cardinality and the number of columns is its degree. For tabular representation of a relation it is customary to name the table and to name each column. The columns (fields) of the table are called attributes. In relational theory the terms referred as "relation", "tuple" and "attributes" are generally referred also as "table", "row" and "columns" (fields) respectively for relational applications. The individual entries in each tuple are called its components. A column or set of columns (attributes) whose values uniquely identify a row of a relation is a candidate key (generally called a key) of the relation. When a relation has more than one field as key, it is conventional to designate one

as the primary key and the other as the secondary key. The term data model represents the complete set of relations stored in the system. A Schema is a set of declarations that describes the data model and a subschema is a set of declarations that describes some part of schema available to a particular user or a group of users belonging to some category. The design of schema and subschema for a database is closely related to the concept of normalization which is primarily based on features of functional dependencies between attributes of a relation.

Functional dependencies are the main features of relational schemes. But it introduces undesirable anomalies in relational scheme, i.e., if an attribute value is updated in one of the relations, its effect on other relations may not be validated on its own. This feature was first recognized by Codd and he suggested the concept of normalized relations to avoid or atleast minimize the effect of above anomaly. The underlying principle of normalization is to replace the initial bigger schemes by several smaller schemes using non-loss decomposition principle as there exists a link between functional dependency and decomposition.

Normalization theory is based on the observation that certain collections of relations have better properties in an updating environment than the other collections

of relations containing the very same data do. The normalization theory then provides a strong foundation for the design of relations which have favourable update properties. The theory is based on series of normal forms (NF) — first (1NF) to third (3NF) and recently to fifth normal form (5NF) — which provides successive improvements in update properties of database and thus minimize insertion, update and deletion anomalies. The most widely known result of normalization theory is third normal form often known as 3 "Boyce-Codd Normal Form" (3BCNF).

The various normal forms can be briefly described as[50]:

First Normal Form:

A relation is in first normal form (1NF), if every component (field) of each tuple (i.e., row of a table) is non decomposable (atomic) i.e., fields must not contain repeating groups.

Second Normal Form:

A relation is in second normal form (2NF) if all non key attributes (fields) are fully dependent on keys i.e., every non key field of first normal form be dependent upon the complete primary key.

Third Normal Form:

A relation is in third normal form (3NF) if all non key attributes are directly and fully dependent on keys. The third normal form (3NF) requires, second normal form tables to be decomposed, if a non key field determines value of another non key field i.e., transitive-dependencies should be eliminated.

Fourth Normal Form:

The fourth normal form (4NF) requires that third normal form (3NF) tables should not contain two or more independent multivalued dependencies.

Fifth Normal Form:

The fifth normal form (5NF) requires that fourth normal form (4NF) tables should not be nonloss decomposed into three or more smaller tables.[54]

In context of relational schemes, nonloss decomposition means that there is no loss of information due to decomposition of a relation into several relations. A non-loss decomposition guarantees that the join produces exactly the same original relation. The technique of nonloss decomposition provides an aid to the relational database design. The basic approach consists of starting with some given relation, together with a statement of certain constraints

in the form of functional dependencies and multi valued dependencies. They are successively decomposed to a collection of relations that are equivalent to the original relation and yet are in some way preferable to it, using the constraints to guide the reduction (decomposition) stages.

Relational schemes giving rise to design of relations (tables) have following major advantages.

Simplicity:

The relational user is presented with a single, consistent data structure. User formulates his query strictly in terms of information content, without any reference to system oriented details.

Data Independence:

According to Date [50] the data independence is defined as "immunity of applications to change in storage structure and access strategy." The relational model makes it possible to eliminate the details of storage structure and access strategy from the user interface.

Symmetry:

Data base systems which are based on connections between records make some questions easier to ask than others viz. questions whose structure matches that of the data base. Since all information is represented by data values in

relations, there is no preferred format for a question at the user interface. It should be noted here that the symmetry of the data model does not necessarily imply symmetry of the associated physical data structures maintained by the system.

Strong Theoretical Foundation:

The relational data model rests on the well developed mathematical theory of relations and on the first order predicate calculus. This theoretical background makes possible the definition of relational completeness, and the rigorous study of good data base design (normalization). A language is said to be relationally complete, if any relation derivable from the given relations (i.e., the database) by means of expression of the relational calculus, can be retrieved using that language.

The other important advantage is the ease of use with which high-level, non-procedural relational languages may be defined. Because they are easy to learn and use, high-level languages make databases available to a new class of casual users who lack the training required by conventional programming languages. High-level languages also give the system maximum flexibility to optimize the execution of a given query and to adapt the stored data structures to the changing needs of the user population. The non-procedural approach to language design permits a unified treatment of data definition,

data manipulation and control. Moreover, high-level languages make it easy to define and manipulate views of data which are not directly supported by physical structures. Though some of the above advantages are common to other data models as well, the database management systems based on the relational data model approach have been the most extensively researched models in the recent years. The reasons are many; the chief among them being their elegance, versatility and power.

5.4 INTERFACE DESIGN:

In order to achieve the data independence and flexible access we first view the input data and output values of standard linear programming problem as a set of normalized relations. Moreover, to capture all essential information pertaining to input data these relations have been designed accordingly. The following set of eight relations are capable of capturing all necessary information related to the input data needed in any linear programming problem.

- (i) `ivar` - the list of variables
- (ii) `icon` - the list of constraints (including the objective function as the first constraint.
- (iii) `ivarn` - the list of variable names
- (iv) `iconsn` - the list of constraint names
- (v) `ivarlb` - the lower bounds (if any)
- (vi) `ivarub` - the upper bounds (if any)

- (vii) imat - the coefficient matrix elements (non-zero values only)
- (viii) iconsd - the constraint detail (type and RHS value)

The structure of these relations is as follows:

- (i) ivar : (Var_id)
- (ii) icon : (Cons_id)
- (iii) ivarn : (var_id⁺, var_name)
- (iv) iconsn : (cons_id⁺, cons_name)
- (v) ivarlb : (var_id⁺, lower_bound)
- (vi) ivarub : (var_id⁺, upper_bound)
- (vii) imat : (cons_id⁺, var_id⁺, mat_element)
- (viii) iconsd : (cons_id⁺, cons_type, rhsvalue)

NOTE: (a) Underlined field(s) constitute the key of the relation.

(b) "+" indicates that the key is linked to a base field.

This linking with the database field leads to minimization of deletion and update anomalies in the relational data base model.

The solution values of the linear program can also be viewed as a set of normalized relations. The following set of four relations are designed to capture all necessary information related to solution of a linear program.

- (i) ovar - solution values
- (ii) ocon - constraint values
- (iii) objr - cost range
- (iv) orhsr - rhs range

The structure of these relations is as follows:

- (i) ovar : (var_id⁺, solution, relative_cost)
- (ii) ocon : (cons_id⁺, slack, dual-price)
- (iii) objr : (var_id⁺, current_cost^{*}, cost-increase,
cost-decrease)
- (iv) orhsr : (cons_id⁺, current_rhs^{*}, rhs_increase,
rhs-decrease)

The type and format of the fields for both the input and solution models are summarized below:

- (a) var_id : string of length 8
- (b) cons_id : string of length 7
- (c) var_name : string of length 32
- (d) cons_name : string of length 32
- (e) all other fields are numeric of float type (13,6)
i.e.,thirteen digits long numbers with six
digits to the right of the decimal point.

NOTE: These formats are compatible with MPS format [92].

Some of the queries necessitate storing the results of a query for which temporary relations i.e."ofunct" and

"orhfunct" have been incorporated in the schema design of interface database. The structure of these relations is as follows:

ofunct : (kofun, ovl, ofl, of2, of3, of4, of5)

orhfunct : (Krhfun, rcl, rfl, rf2, rf3, rf4, rf5)

Using proposed schemes for a set of input relations as well as a set of output relations the actual design of relational database is done by UNIFY DBMS package implemented on UNIX based machine UPTRON S-32 at IIT Kanpur. Further the problem solver LINDO is used to solve the model formulation done by modelling language component of LAMP. For an introductory precise reference to novice users, the specific features of the problem solver LINDO is given in Appendix D. The pertinent information about UNIFY package for user acquaintance is given in Appendix E. The salient details of the data manipulation language ~~embedded~~^{attached} to UNIFY DBMS package (i.e., SQL) is given in Appendix F.

Once the solution to model formulation is done by LINDO, several sets of interface programs transform the model formulation data as well as solution data in a format acceptable to the database (i.e., in accordance with schemes proposed above). Using any appropriate utility to enter the data to database, the data entry to relations is achieved. The schema report listing for input relations as proposed is given

in Figure 6. The schema report listing for the output associated relations is given in Figure 7. The set of input and output relations data generated for agricultural planning sample problem, as described in Section 4.4.1 is shown in Figure 8 and Figure 9 respectively. Thus the interface module has been suitably designed to capture all relevant information necessary for a linear programming model. In the next chapter we discuss the detailed data analysis capabilities of LAMP.

* !! NOTE : Lines starting with '*' are 'Comment' lines.

* INPUT RELATIONS

SCHEMA REPORTS

Schema Listings

```

* -----
RECORD/FIELD      REF      TYPE      LEN      LONG NAME

ivar              100                      var_list
    *kivar                      STRING    8      var_id
icon              100                      cons_list
    *kicon                      STRING    7      cons_id
ivarn             100                      var_name_list
    *kivarn                     kivar    STRING    8      var_id
    varname                   STRING   32      var_name
iconsn            100                      cons_name_list
    *kiconsn                   kicon    STRING    7      cons_id
    consname                  STRING   32      cons_name
ivarlb            50                      var_lower_bound
    *kvarlb                     kivar    STRING    8      var_id
    lb                        FLOAT   136      lower_bound
ivarub            50                      var_upper_bound
    *kvarub                     kivar    STRING    8      var_id
    ub                        FLOAT   136      upper_bound
iconsd            100                      cons_detail
    *kiconsd                   kicon    STRING    7      cons_id
    type                      STRING    1      cons_type
    rhval                     FLOAT   136      rhsvalue
imat              500                      matrix
    *kimat                     kiconm    STRING    7      cons_id
    kiconm                   kivar    STRING    8      var_id
    kivar                     kivar    STRING    8      var_id
    matval                    FLOAT   136      mat_element

```

* -----
Note: Starred field(s) indicate(s) key(s) of relations.

Fig. 6: Schema Listing for Input Relations.

* !! NOTE : Lines starting with '*' are 'Comment' lines.

* OUTPUT RELATIONS

SCHEMA REPORTS

Schema Listing

```

* -----
RECORD/FIELD      REF      TYPE      LEN      LONG NAME

over              100
*kover            kivar      STRING     8      var_value
oval              FLOAT     136      var_id
rcost              FLOAT     136      solution
                  relative_cost

ocon              100
*kcon              kicon      STRING     7      cons_value
oval              FLOAT     136      cons_id
dprice             FLOAT     136      slack
                  dual_price

obur              100
*kobu              kibu      COMP
kobur              kivar      STRING     8      cost_range
ccof              FLOAT     136      var_id
ocai              FLOAT     136      current_cost
ocad              FLOAT     136      cost_increase
                  cost_decrease

orhsr              100
*korhs              COMP      korhs
                  rhs_range

kforhsr            kicon      STRING     7      cons_id
oval              FLOAT     136      current_rhs
rhval1             FLOAT     136      rhs_increase
rhval2             FLOAT     136      rhs_decrease

ofunct            100
*kofun              kofun      STRING     8      ofunct
of1                STRING     8      of1
of2                FLOAT     136      of1
of3                FLOAT     136      of2
of4                FLOAT     136      of3
of5                FLOAT     136      of4
                  of5

orhfunct           100
*krhfun              krhfun      STRING     7      orhfunct
rc1                STRING     7      rc1
rf1                FLOAT     136      rf1
rf2                FLOAT     136      rf2
rf3                FLOAT     136      rf3
rf4                FLOAT     136      rf4
rf5                FLOAT     136      rf5

```

* -----
note: Starred field(s) indicate(s) key(s) of relations.

Fig. 7: Schema Listing for Output Relations.

* !! NOTE : Lines starting with '*' are 'Comment' lines.

* SAMPLE INPUT RELATIONS FOR AGRICULTURAL MODEL
* *****

var_id

SHEEP
CATTLE
ORANGE
APPLE
ONION
COTTON

cons_id

21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

cons_id:cons_name

21	:CEIL	SHEEP
20	:CEIL	CATTLE
19	:CEIL	ORANGE
18	:CEIL	APPLE
17	:CEIL	ONION
16	:CEIL	COTTON
15	:FERTAIL	JULY
14	:FERTAIL	JUNE
13	:FERTAIL	MAY
12	:REYIELD	
11	:WATERAIL	JULY
10	:WATERAIL	JUNE
9	:WATERAIL	MAY
8	:ORCHLAND	
7	:FIELDLAND	
6	:TOTLAND	
5	:LABORAIL	
4	:TOTWATER	JULY
3	:TOTWATER	JUNE
2	:TOTWATER	MAY

Fig. 9: Sample Input Relations for Agricultural Model, (Contd.).

* !! NOTE : Lines starting with '*' are 'Comment' lines.
 * SAMPLE INPUT RELATIONS FOR AGRICULTURAL MODEL(continued)
 * *****

cons_id	cons_type	rhsvalue
21	!L	300.00000
20	!L	400.00000
19	!L	800.00000
18	!L	500.00000
17	!L	250.00000
16	!L	2000.00000
15	!L	6000.00000
14	!L	5000.00000
13	!L	4000.00000
12	!G	15000.00000
11	!L	269999.99881
10	!L	259999.99940
9	!L	200000.00000
8	!L	600.00000
7	!L	1849.99999
6	!L	2699.99999
5	!L	5850.00002
4	!L	275000.00000
3	!L	265000.00060
2	!L	205000.00119

cons_id	var_id	mat_element
21	!SHEEP	1.00000
4	!SHEEP	0.60000
3	!SHEEP	0.50000
2	!SHEEP	0.40000
1	!SHEEP	719.99998
20	!CATTLE	1.00000
4	!CATTLE	0.30000
3	!CATTLE	0.20000
2	!CATTLE	0.10000
1	!CATTLE	600.00000
19	!ORANGE	1.00000
15	!ORANGE	12.00000
14	!ORANGE	10.99999
13	!ORANGE	10.00000
12	!ORANGE	15.00000
11	!ORANGE	85.00000
10	!ORANGE	75.00000
8	!ORANGE	1.00000
6	!ORANGE	1.00000
5	!ORANGE	1.50000
4	!ORANGE	85.00000

Fig. 8: Sample Input Relations for Agricultural Model, (Contd.).

* !! NOTE : Lines starting with '*' are 'Comment' lines.

* SAMPLE INPUT RELATIONS FOR AGRICULTURAL MODEL ... (continued)
 * *****

cons_id	var_id	mat_element
3	:ORANGE	75.00000
1	:ORANGE	8812.99996
18	:APPLE	1.00000
15	:APPLE	9.00000
14	:APPLE	8.00000
13	:APPLE	7.00000
12	:APPLE	14.00000
11	:APPLE	63.99999
10	:APPLE	50.00000
8	:APPLE	1.00000
6	:APPLE	1.00000
5	:APPLE	1.00000
4	:APPLE	63.99999
3	:APPLE	50.00000
1	:APPLE	4814.00013
17	:ONION	1.00000
15	:ONION	6.00000
14	:ONION	5.00000
13	:ONION	4.00000
12	:ONION	12.99999
10	:ONION	60.00000

cons_id	var_id	mat_element
7	:ONION	1.00000
6	:ONION	1.00000
5	:ONION	2.69999
3	:ONION	60.00000
1	:ONION	6110.00001
18	:COTTON	1.00000
15	:COTTON	3.00000
14	:COTTON	2.00000
13	:COTTON	1.00000
12	:COTTON	12.00000
11	:COTTON	90.00000
10	:COTTON	80.00000
9	:COTTON	65.00000
7	:COTTON	1.00000
6	:COTTON	1.00000
5	:COTTON	2.90000
4	:COTTON	90.00000
3	:COTTON	80.00000
2	:COTTON	65.00000
1	:COTTON	6453.00007

Fig. 8: Sample Input Relations for Agricultural Model.

* NOTE : Lines starting with '*' are 'Comment' lines.

* SAMPLE OUTPUT RELATIONS FOR AGRICULTURAL MODEL

* *****

var_id :	solution:	relative_cost
SHEEP :	300.00000:	0.00000
CATTLE :	400.00000:	0.00000
ORANGE :	37.50000:	0.00000
APPLE :	0.00000:	1795.74990
ONION :	0.00000:	2546.25010
COTTON :	1849.99999:	0.00000

cons_id:	slack:	dual_price
21 :	0.00000:	719.99998
20 :	0.00000:	600.00000
19 :	762.50000:	0.00000
18 :	500.00000:	0.00000
17 :	250.00000:	0.00000
16 :	150.00000:	0.00000
15 :	0.00000:	734.41666
14 :	887.50034:	0.00000
13 :	1775.00000:	0.00000
12 :	7762.49981:	0.00000
11 :	100312.50268:	0.00000
10 :	109187.50316:	0.00000
9 :	79750.00203:	0.00000
8 :	562.50000:	0.00000
7 :	0.00000:	4249.75014
6 :	812.50000:	0.00000
5 :	428.74999:	0.00000
4 :	105012.50327:	0.00000
3 :	113957.50344:	0.00000
2 :	84590.00230:	0.00000

var_id :	current_cost:	cost_increase:	cost_decrease
SHEEP :	719.99998:	999998.99864:	719.99998
CATTLE :	600.00000:	999998.99864:	600.00000
ORANGE :	887.50034:	16999.00001:	2394.33330
APPLE :	4214.00013:	1795.74990:	999998.99864
ONION :	6110.00000:	2546.25010:	999998.99864
COTTON :	6453.00007:	999998.99864:	2546.25010

cons_id:	current_rhs:	rhs_increase:	rhs_decrease
21 :	300.00000:	175020.80033:	300.00000
20 :	400.00000:	350041.60067:	400.00000
19 :	600.00000:	999998.99864:	762.50000
18 :	500.00000:	999998.99864:	500.00000
17 :	250.00000:	999998.99864:	250.00000
16 :	2000.00000:	999998.99864:	150.00002
15 :	6000.00000:	966.18311:	450.00000
14 :	5000.00000:	999998.99864:	887.50034
13 :	4000.00000:	999998.99864:	1775.00000
12 :	15000.00000:	7762.49981:	999998.99864
11 :	269999.99881:	999998.99864:	100312.50268
10 :	259999.99940:	999998.99864:	109187.50316
9 :	200000.00000:	999998.99864:	79750.00203
8 :	600.00000:	999998.99864:	562.50000
7 :	1849.99999:	150.00000:	940.90507
6 :	2699.99999:	999998.99864:	812.50000
5 :	5250.00002:	999998.99864:	428.74999
4 :	275000.00000:	999998.99864:	105012.50327
3 :	285000.00000:	999998.99864:	113957.50344
2 :	205000.00119:	999998.99864:	84590.00230

* *****

END

Fig. 9: Sample Output Relations for Agricultural Model.

CHAPTER VI

DATA MANAGEMENT ASPECTS OF LAMP

6.1 INTRODUCTION:

Most real life LPP's tend to be of reasonably large size involving several thousands of variables and several hundreds of constraints. Obviously, one has to ensure that each and every coefficient that goes into the model is correct both in its value and its association with appropriate variable and constraint. This is definitely not a trivial issue. The early matrix generators used several utilities to perform this task. Routines for the above basic task have been incorporated in PERUSE [110] and sophisticated testing routines to reveal common errors have been incorporated in ANALYZE [82]. Substantial portion of mathematical programming software like MPSX [92], LINDO [145] is devoted to the job of data validation. Data validation which is the part of model validation ensures the exact value of data used in algorithmic form in place of coefficients and right-hand-side values in the form of identifiers and parameters of the modeller's form.

Modelling languages incorporate distinction between symbolic and explicit forms of data and thereby offer more

flexible and more convenient management of data than the common matrix-generation language. Symbolic representation of data also promotes reliability by serving as a check on validity of explicit data. The modelling language translator can detect explicit data that are not specified in conformance with the modelling language symbolic representation.

The modeller needs help in the presentation of results which again become voluminous. Report-writer programs devoted to this task have been appended to the standard mathematical programming software. The LP solver generates an optimal basis. But the modeller/user wants to know the optimal activities and other solution values. Thus any linear programming system needs ways of storing and retrieving both input data and solution values, extracting solution data to make readable reports. Both the tasks viz. data validation and report writing act as heavy overheads on the problem solving software. Based on the solution of the model, usually the modeller may have some query. In addition, detailed "what if" analyses that are mandatory in any modelling activity require an ability to modify the data and the structure and re-solve the problem, interactively.

A modelling language provides the modeller with a medium to express the model structure to a model manage-

ment system, which in turn generates the detailed model data for the problem processor. To support the modeller in the interactive editing, report writing and "what if" analysis stages we need a similar medium for expressing the model manipulation function. We initially planned to design a language that provides model manipulation capability. A closer look, however, revealed the fact that the manipulation of the structure can be handled using the model language itself in association with a standard text editor. Other manipulations primarily amount to data manipulation and the well developed theory of relational databases and a query language based on a relational system should be able to provide this capability directly. Of course, this would mean an ability to interface modelling language with a query language. The relational view of the linear programming problem precisely provide such an interface. We ^{implemented} ~~planned~~ the idea by interfacing LAMP with a relational system UNIFY [168] which supports a query language SQL [95].

It turns out that our conjecture is in fact true, as we demonstrate it in detail in the following sections. Thus we managed to exploit the rich capabilities of the

query language without designing another language for model manipulation. Additionally the ideas we demonstrate are general enough to be of value in other application areas as well.

6.2 DATA ANALYSIS:

6.2.1 Introduction:

A relational view of the input data and solution values enables us to exploit the power of the relational DBMS software to assist the analyst in the different aspects of the modelling process. Various kinds of "sensitivity" and "what if" analyses can be performed in a flexible manner. Every relational DBMS software supports a query language which is non-procedural. Non-procedurality allows the analyst/modeller to view the task without getting involved much in the details of how exactly the task must be translated into a set of procedures. The actual procedure conversion can be conveniently left to the machine, thereby permitting a meta-level interface between the modeller and the machine. Most of the query languages have the property of relational completeness i.e., any query of arbitrary complexity concerning the database can be written with the help of the query languages. Also most query languages support user friendly interfaces for interactive editing, report writing, data entry, etc. Such an environment facilitates fairly complex analysis, to be performed with ease.

UNIFY, a relational database management system software which has been implemented for data management aspect uses Structured Query Language (SQL). SQL permits "select", "from", "where" constructs for simple queries. It further allows nesting, arithmetic computations, delete, insertion and update features to build desired queries of varying complexities. UNIFY also has embedded "SQL by Forms" based on the Query-By-Example (QBE) approach. In addition it has a listing processor (LST) and a report processor (RPT) which can present and produce formatted reports based on results of query as desired by an analyst/user. Typically most of the current generation relational DBMS provide similar capabilities.

SQL is emerging to be one of the most popular query languages. Data analysis for various information related to real world problem model can be done using SQL. We illustrate this idea in following subsections. For the sake of clarity we classify the analysis into three stages viz.,

- (i) Input Data Analysis
- (ii) Solution Report
- (iii) "What-if" Analysis

6.2.2 Input Data Analysis:

The modeller/analyst may like to verify all possible coefficient and right-hand-side values that go into the

model as input data. Further, based on them he may like to have information pertaining to problem statistics, problem summary, simple error checking, consistency checking and sophisticated checking. Each of them is discussed separately.

(i) Problem Statistics:

Statistics like the count of variables, count of constraints of different type, nonzero coefficient count, density of the matrix, etc. can be conveniently obtained using SQL. Such gross measures provide a preliminary level of check and a rough idea of complexity of the numerical solution of the problem. Sample query (Q:1) in Figure 10 illustrates these features.

In the first query (Q:1.1) the "select" verb uses aggregate function "count*" to obtain the total number of records (i.e., number of variables) from the input relation var_list (ivar).

Similarly the second query (Q : 1.2) finds the total number of records (i.e., number of constraints) from the input relation cons_list (icon).

In the third query (Q:1.3) the "select" verb obtains the total number of such records whose constraint type is less than equal to ("< =" or "L") in the input relation cons_detail (iconsd). The "where" clause specifies the required condition.

* !! NOTE : Lines starting with '*' are 'Comment' lines.

```

=====

*      { Q:1 ... Problem statistics }
*      { ----- }

*      { Q:1.1 ...No of variables}

              select count(*) from iver/

*      { Q:1.2 ...No of constraints}

              select count(*) from 1con /

*      { Q:1.3 ...No of = constraints }

              select count(*) from 1consd where cons_type='L'/

*      { Q:1.4 ...No of nonzero coefficients}

              select count(*) from 1mat /

*      { Q:1.5 ...No of unit coefficients}

              select count(*) from 1mat where 1mat.mat_element=1'

*      { Q:1.6 ... Matrix density }

              delete ofunct'

              insert into ofunction: 'matden'

              update ofunct set of1= select count(*) from 1mat:
              where kofun='matden*'/

              update ofunct set of2= select count(*) from 1ver:
              where kofun='matden*'/

              update ofunct set of3= select count(*) from 1con:
              where kofun='matden*'/

              update ofunct set of4=of1//of2*of3)
              where kofun='matden*'/

              select kofun,of1,of2,of3,of4 from ofunct
              where kofun = 'matden*' /

=====

```

Fig. 10: Queries for Problem Statistics.

The fourth query (Q:1.4) obtains the total number of records from input relation matrix (imat). This will be the total number of nonzero coefficients as input-relation matrix (imat) initially stores nonzero coefficient values only.

In the fifth query (Q: 1.5) the "select" verb obtains the total number of specific records from input relation matrix (imat), using "where" clause specifying the required condition (i.e., only those records for which the coefficient values are unity).

The query set (Q:1.6) uses several queries to find the matrix density of LP matrix stored in the data base. In order to compute the matrix density the output relation (ofunct) is initialized by deleting all its previous records and fields. The key (kofun) of relation (ofunct) is set as "matden*" . Then the field "ofunct.of1" for record having key as "matden*" is set equal to the value obtained from another query block which finds the total number of nonzero coefficients in the input relation matrix (imat). Similarly the field "ofunct.of2" is set equal to total number of variables for an input record with key as "matden*". Next query block updates the field "ofunct.of3" with a value equal to total number of constraints for an input record with key as "matden*". Next query block updates the field "ofunct. of4" with a value equal to required matrix density which is computed using previously stored values corresponding to an input record with key as "matden*" .

Finally the result is displayed by selecting various fields and result from the relation (ofunct) having key as "matden*".

Thus Figure 10 illustrates queries representing "Problem statistics" features associated with a linear program.

(ii) Problem Summary:

Sometimes the analyst/modeller may be interested in input data summary only. An ordered list (in any specific order) of variables, constraints, variable names, constraint names, right-hand-sides, bounds, smallest and largest elements in matrix, etc. can be obtained to enable a neat summary of the problem data. Sample query (Q:2) in Figure 11 illustrates these features.

In particular, in the first query (Q:2.1) the "select" verb obtains all the constraint names (cons_name) from input relation cons_name_list (iconsn) and displays them in ascending order of constraint names.

In the second query (Q:2.2) all the fields of input relation var_upper_bound (ivarub) is displayed in descending order of upper_bound values.

In the third query (Q:2.3) the first query block obtains the smallest coefficient value from input relation matrix (imat) and the second query block obtains the largest coefficient value from the same input relation matrix (imat).

```

*  !! NOTE : Lines starting with '*' are 'Comment' lines.

*=====
*      { Q:2 ... Problem summary }
*      { ----- }

*      { Q:2.1 ... List of constraint names in ascending order }
              select cons_name
              from iconstr
              order by cons_name asc/

*      { Q:2.2 ... List of bounds in descending order }
              select *
              from iverub
              order by upper_bound desc/

*      { Q:2.3 ...Smallest and largest element in matrix }
              select min(mat_element)
              from mat
              where mat_element < 0/
              select max(mat_element)
              from mat
              where mat_element > 0/
*=====

```

Fig. 11: Queries for Problem Summary.

Thus Figure 11 provides sample queries representing "Problem Summary" features associated with a linear program.

(iii) Simple Error Checking/Consistency Checking:

Model validation requires model to represent the real life problem exactly. The input data values should be properly associated with variables with reference to a specific constraint. Minimal consistency checks are provided by the relational model directly. For example, the linking of the key of the input relation matrix (imat) with the key (kivar) of var_list (ivar) and key (kicon) of cons_list (icon) automatically ensures that the matrix elements are defined only for valid variables and constraints that are already available in (i.e., known to) the database. Checking for a variable that does not have nonzero coefficient in any constraint and/or a constraint with no nonzero coefficient corresponding to any variable can provide a simple error check. Such errors easily creep in many of the large scale problems due to a variety of reasons. Also one can easily check for any constraint(s) with all zero/negative coefficient with a positive right-hand-side (RHS) element, which indicates a serious inconsistency. Sample query (Q:3) in Figure 12 illustrates these features.

In the first query (Q:3.1) the inner query block first obtains the list of variable identifiers (var_id) from input relation matrix (imat) for which the coefficient value


```

*  ' NOTE : Lines starting with '*' are 'Comment' lines.

*=====
*      { Q:3 ... Simple error checks}
*      { -----}

*      { Q:3.1 ... Check for variable with no nonzero matrix coefficient }
              select ivar.var_id from ivar where ivar.var_id !=
              select imat.var_id from imat where imat.mat_element != 0

*      { Q:3.2 ... Check for constraints having no nonzero LHS elements but
              with nonzero RHS Values }
              select icon.cons_id,cons_type,rhsvalue from icon,iconsd
              where icon.cons_id=iconsd.cons_id and rhsvalue = 0
              and iconsd.cons_id !=
              select imat.cons_id from imat
              where imat.mat_element != 0 and imat.cons_id != '1
*=====

```

Fig. 12: Queries for Simple Error Checking.

(mat_element) is not zero. Then the outer query selects the variable identifiers (var_id) from input relation var_list (ivar) which are not contained (if any) in the list of variables obtained from the inner query. Thus the entire query block is able to check for a variable with no nonzero matrix coefficient.

In the second query (Q:3.2) the inner query block obtains from input relation matrix (imat) all those constraint identifiers (imat.cons_id) other than objective function, and for which the coefficient value(s) (mat_element) is(are) not equal to zero. Then the outer query block selects from input relations cons_list and cons_detail only those constraint identifiers which are not in the set of constraint identifiers obtained as the result of inner query block. Thus the inner query block selects all constraints other than objective function which have coefficients for left-hand-side (LHS) and then checks for non-match with constraint identifiers (cons_id) of relation cons_detail having right-hand-side value (rhs value) other than zero. In the above manner the query provides simple error check for constraints having no nonzero LHS elements but with non-zero RHS value.

Thus Figure 12 provides sample queries representing "Simple error checking" features associated with a linear program.

(iv) Sophisticated Checking:

Many of the LP packages provide for a picture clause which pictures (displays) elements in a range say (100-999). One can easily get such pictures using SQL. Problem specific data checking like the set of variables that use specific resource, say "WATER" (in our example agricultural planning model), can be obtained. This kind of checking may be quite significant for model validation as a quick check. Relatively easier checks for simple upper bounds, generalized upper bounds can also be made. Pattern matching, say checking for a "transportation" type constraint set, can also be obtained. Sample query (Q:4) in Figure 13 illustrates these features.

The first query (Q:4.1) obtains all fields of input relation matrix (imat) for which the coefficient values (mat_element) lie between 100 and 999 and displays the result in ascending order of fields (var_id) and (cons_id) of relation matrix (imat).

In the second query (Q:4.2) the "select" verb obtains variable identifiers (var_id) uniquely from input relations matrix (imat) and cons_name_list (iconsn) for which constraint identifiers are same and constraint name (cons_name) is having a match either with string "WATER*" or with string "*WATER*". Thus the above query is able to obtain the list of variables that use a specific resource, say "WATER" in this case.

```

*  ' NOTE : Lines starting with '*' are 'Comment' lines.

*=====
*      { Q:4 ... Sophisticated checking }
*      { ----- }
*      { Q:4.1 ... Picturing the matrix in a range }
          select * from imat where imat.mat_element between 100
              and 999
          order by imat.var_id asc, imat.cons_id asc/

*      { Q:4.2 ... Variables that use a specific resource - say WATER }
          select unique imat.var_id, cons_name from imat, iconsn
          where imat.cons_id=iconsn.cons_id and cons_name is in
              'WATER', 'WATER*' /

*      { Q:4.3 ... Simple Upper Bound }
          select iconsn.cons_id, cons_name from iconsn, imat
          where imat.cons_id=iconsn.cons_id and iconsn.cons_id <=
          select imat.cons_id from imat where imat.mat_element <= 1;
          group by iconsn.cons_id
          having sum(imat.mat_element) <= 1

*      { Q:4.4 ... Guaranteed Upper Bound }
          select iconsn.cons_id, cons_name from iconsn, imat
          where imat.cons_id=iconsn.cons_id and iconsn.cons_id <=
          select imat.cons_id from imat where imat.mat_element <= 1;
          group by iconsn.cons_id
          having sum(imat.mat_element) <= 1/

*      { Q:4.5 ... Transportation type pattern }
          select imat.var_id, count(*) from imat
          where imat.mat_element=1 and imat.cons_id<='1'
          group by imat.var_id /
          select imat.cons_id, count(*) from imat
          where imat.mat_element=1 and imat.cons_id<='1'
          group by imat.cons_id'
*=====

```

Fig. 13: Queries for Sophisticated Checking.

In the third query (Q:4.3) the inner query block obtains the matrix (imat) constraint identifiers (cons_id) for which the coefficient value (mat_element) is not unity. Then the outer query block selects only those constraint identifiers and associated constraint names from the join of input relations cons_name_list (iconsn) and matrix (imat) which do not match with the results of inner query and finally displays after grouping them according to constraint identifier for which the coefficient value (mat_element) sum is equal to unity only. Thus the above query detects the simple upper bound situation.

Similarly, in the fourth query, (Q:4.4) the inner query block obtains those matrix (imat) constraint identifiers (cons_id) for which the coefficient value (mat_element) is not unity. Then the outer query block selects only those constraint identifiers and associated constraint names from the join of input relations cons_name_list (iconsn) and matrix (imat), which do not match with the results of inner query and finally displays after grouping them according to constraint identifier for which the coefficient value (mat_element) sum is greater than unity. Thus the above query is able to display the generalized upper bound situation.

In the last set of two queries (Q:4.5) the first query obtains from input relation matrix (imat) all those variable identifiers (var_id) for which

the coefficient value (`mat_element`) is equal to unity and constraint identifier (`cons_id`) is other than for objective function. Then after grouping according to variable identifier, displays the variable identifier alongwith total number of such occurrences in that group. The second query obtains all those matrix constraint identifiers (`imat.cons_id`) for which the coefficient value (`mat_element`) is equal to unity and constraint identifier (`cons_id`) is other than for objective function. Then after grouping according to constraint identifier, displays the constraint identifier along with total number of such occurrences in that group. In this way above query can detect "transportation" type pattern.

Thus Figure 13 provides sample queries representing "sophisticated checking" features associated with a linear program.

6.2.3 Output Analysis:

The analyst/modeller would not stop at model construction and input data analysis. He may generally prefer to go for solution values and further related analysis based on the solution obtained from the problem solver. In particular, he may like to have solution analysis or some kind of sophisticated analysis.

(i) Solution Analysis:

The modeller/analyst may be interested in some simple report or partial summary related to solution values e.g.,

printing of solution values in a specified order, partial summary like breakup of profit into broad categories, printing of constraints in a specified order, say decreasing order of shadow prices. Sample query (Q:5) in Figure 14 illustrates these features.

The first query (Q:5.1) obtains the output variable identifiers and associated solution values from output relation `var_value` (`ovar`) for which the solution value is greater than zero and finally displays the result in descending order of solution values.

In the second query (Q:5.2) the first query block obtains the profit contribution from atoms of the "ORCHARD" molecule-using aggregate function "sum". This is achieved by selecting the solution value from output relation `var_value` (`ovar`) and profit coefficient value from input relation matrix (`imat`) corresponding to same variable identifier and taking product of the two values. The products are added together using aggregate function "sum". Similarly second query block obtains the profit contribution coming from the field (i.e., "FIELD") molecule atoms (i.e., "COTTON", "ONION"). Thus the query displays the profit, contribution from the specified type of crops in the current solution.

Thus Figure 14 illustrates queries representing "Solution analysis" features associated with a linear program.

* !! NOTE : Lines starting with '*' are 'Comment' lines.

*=====

* { Q:5 ... Solution Analysis }

* { ----- }

* { Q:5.1 ... Simple Report }

select over.var_id,over.solution from over

where over.solution > 0

order by over.solution desc /

* { Q:5.2 ... Partial Summary - Contribution from Orchard and Field }

select sum(imat.mat_element*over.solution) from imat,over

where imat.var_id=over.var_id and imat.cons_id='1'

and over.var_id is in ('APPLE*','ORANGE*') /

select sum(imat.mat_element*over.solution) from imat,over

where imat.var_id=over.var_id and imat.cons_id='1'

and over.var_id is in ('COTTON*','ONION*') /

select * from imat where imat.cons_id = '1'

select * from over /

*=====

Fig. 14: Queries for Solution Analysis.

(ii) Sophisticated Analysis:

The analyst may like to have specialized reports such as list of variable identifiers in increasing order of profit contribution and the list of binding constraints along with the right-hand-side value for each of such constraint identifiers. Sample query (Q.6) illustrates these features in Figure 15.

The first query (Q:6.1) displays the product of coefficient value corresponding to objective function constraint identifier and a variable identifier from input relation matrix (imat) and "solution" value corresponding to some variable identifier from output relation var_value (ovar) for which the solution value is greater than zero only. The result is displayed in default (ascending) order of the product value.

The second query (Q:6.2) selects those constraint identifiers for which slack value is zero. This is done by joining the relation (ocon) and (iconsd) using constant identifier (cons_id). In the display the constant identifier and rhs value are shown.

This Figure 15 illustrates queries representing "sophisticated analysis" features associated with a linear program.

```

*  !! NOTE : Lines starting with '*' are 'Comment' lines.

*=====
*      { Q:6 ... Sophisticated Analysis }
*      / ----- }

*      { Q:6.1 ... Variable List in increasing order of profit contribution. }

      select  imat.mat_element*ovar.solution from imat,ovar
      where imat.var_id=ovar.var_id and imat.cons_id='1'
      and over.solution > 0 /

*      { Q:6.2 ... Binding constraint with RHS values }

      select ocon.cons_id, iconsd.rhsvalue
      from ocon, iconsd
      where ocon.cons_id = iconsd.cons_id
      and ocon.elect = 0;
      order by ocon.cons_id desc

*=====

```

Fig. 15: Queries for Sophisticated Analysis.

6.2.4 "What-if" Analysis:

Once the solution for the problem is known, the analyst/modeller is further interested to go for several "what-if" analyses related to problem domain. The major ones are feasibility check, routine sensitivity analysis, 100% rule, simple analysis, revised solution with some perturbation, relative ratio of contribution and effect of tightening/relaxing an inequality. Sample query Q:7 illustrates these features in Figure 16. Each one of them is discussed below.

(a) Feasibility Check:

In an LP model any initial solution is of much significance to start with. A modeller may be interested in feasibility check of a specified solution or a perturbed solution, i.e., he may be interested to check feasibility if some variables are increased by certain percentage while some other variables are decreased by a certain percentage. This kind of feasibility check can also be made. The query (Q:7.1) illustrates this feature in Figure 16.

The query (Q:7.1) displays for each constraint identifier the detailed information viz. constraint identifier, sum of the product of coefficient value and associated variable solution value, the constraint type and right-hand-side value from relations cons_detail, matrix and var_value. In this process care is taken to ensure that constraint identifier

* !! NOTE : Lines starting with '*' are 'Comment' lines.

```

=====
{ Q:7 ... What If Analysis }

{ Q:7.1 ... Feasibility Check }

    select iconsd.cons_id,sum(imat.mat_element*ovar.solution),
           cons_type,rhsvalue from iconsd,imat,ovar
    where iconsd.cons_id = imat.cons_id and over.var_id=
           imat.var_id and imat.cons_id <= '1'
    group by imat.cons_id/

{ Q:7.2 ... Routine Sensitivity Analysis : Output in some order }

    select sum(ovar.solution*current_cost) from over,obur
    where over.var_id=obur.var_id and over.solution > 0
    select over.var_id,ovar.solution,over.solution*current_cost
    from over,obur
    where over.var_id = obur.var_id
    order by current_cost desc

{ Q:7.3 ... 100% Rule : Simultaneous Changes within Ranges }

{ All variable objective coefficients change by 2% }

    delete ofunct/
    lines 0
    select obur.var_id,obur.var_id,current_cost,cost_increase+
           cost_decrease,current_cost*1.02 from obur
    into f1/
    insert into ofunct(ofun,cv1,of1,cf2,cf3,cf4):
    from f1/
    update ofunct set of5=(of4-of1)/cf2 where (cf4-of1) < 0
    update ofunct set of5=(of4-of1)/cf3 where (of4-of1) > 0 ,
    select * from ofunct/

    select sum(of5) from ofunct,ovar

    where ofun= over.var_id /

{ Q:7.4 ... Simple Analysis }

{ Delete a variable say 'ONION*' & create MPS file }

    lines 0
    separator ' '
    select cons_type,iconsd.cons_id from iconsd into w2/

    lines 0
    separator ' '
    select imat.var_id,imat.cons_id,imat.mat_element from imat

    where imat.var_id <= 'ONION*' into w4/

    lines 0
    separator ' '
    select iconsd.cons_id,rhsvalue from iconsd into w6/
=====

```

Fig. 16: Queries for "What-If" Analysis, (Contd.).

* !! NOTE : Lines starting with '*' are 'Comment' lines.

```

=====
{ Q:7.5 ... Revised Solution with some Perturbation }
delete ofunct/
lines 0
select over.var_id,over.var_id,over.solution,relative_cost
from over into f2/
insert into ofunct(kofun,ov1,of1,of2):
from f2/
update ofunct set of4 = of1*0.97 where kofun is in
'COTTON','ONION'//
update ofunct set of4=of1*1.02 where not kofun is in
'COTTON','ONION' /
select * from ofunct/
select iconsd.cons_id,sum(imat.mat_element*of4),
cons_type,rhsvalue from iconsd,imat,over,ofunct
where iconsd.cons_id=imat.cons_id and over.var_id=imat.var_id
and kofun = imat.var_id and imat.cons_id = '1'
group by imat.cons_id/

{ Q:7.6 ... Relative ratio of contribution from crops and livestock
as function of cost vector }
{ All Basic variable objective coefficients change by 2% }

delete ofunct/
insert into ofunct(kofun,ov1,of1,of2,of3):
select objr.var_id,objr.var_id,current_cost,
cost_increase,cost_decrease from objr/
update ofunct set of4 =
select over.solution from over, ofunct
where over.var_id=kofun/
update ofunct set of5= 1.02*of1*of4/
insert into ofunct(kofun,ov1,of1,of2,of3,of4,of5):
'result','result',0.0,0.0,0.0,0.0,0.0 /
update ofunct set of5=
select sum(of5) from ofunct
where kofun='result':
where kofun = 'result' ,
update ofunct set of1=
select sum(of5) from ofunct where kofun='result' and
kofun is in 'COTTON','ONION' ;
where kofun= 'result'//

update ofunct set of2=
select sum(of5) from ofunct where kofun='result' and
not kofun is in 'COTTON','ONION'//
where kofun = 'result'//

select kofun,of1,of2,of5.of1/of5,of2/of5 from ofunct
where kofun = 'result'//

{ Q:7.7 ... Changing the inequalities to equality for a set of constraints }
select * from iconsd /
update iconsd set cons_type = 'E'
where iconsd.cons_id is in '5','6','7' /

select * from iconsd /
=====

```

Fig. 16: Queries for "What-If" Analysis.

does not belong to objective function; solution values are greater than zero; constraint identifiers of relation, "cons_detail" and "matrix" match and variable identifiers of relations "matrix" and "var_value" also match. Thus from the display the user can check for the validation/violation of feasibility of solution values.

(b) Routine Sensitivity Analysis

The modeller may be interested in some kind of sensitivity analysis done on routine basis viz. displaying output in some order. The query (Q.7.2) illustrates this feature in Figure 16.

In the sample query (Q : 7.2) the first block obtains sum of the product of variable "solution" value and associated current_cost from relation "var_value" and "cost_range". This is conditioned by the following variable identifiers of relations "var_value" and "cost_range" should match, and variable "solution" value is greater than zero. The second block of query then selects variable identifiers corresponding to var_value, "solution" value and product of "solution" value and current_cost from relations var_value and cost_range. In the process care is taken to ensure that variable identifiers of relations var_value and cost_range match. Finally, the result is displayed in descending order of current_cost.

(c) 100% Rule:

All the range analysis traditionally is based on single change in either objective function coefficient or right-hand-side (RHS) value. However, for simultaneous change within range in more than one value of either objective function coefficient or right-hand-side (RHS) value "100% Rule" [31] can be tested. The query (Q:7.3) illustrates this feature in Figure 16.

The sample query (Q:7.3) checks the "100 percent Rule" for 2% increase in basic variables values. In order to test this rule the query block deletes the output relation "ofunct" and then initializes the fields of "ofunct" and it sets field "of5" of relation "ofunct" equal to absolute proportion of contribution from each updated variable. The next query displays fields of relation "ofunct". Further query block obtains the value of summation for all records for which the "ofunct" key matches with variable identifier of relation var_list.

(d) Simple Analysis:

A modeller/analyst may like to add or delete one or more variable (s) and sometimes prefer to add/delete one or more constraint (s). Sample query (Q:7.4) illustrates one such situation in Figure 16. Simple analysis such as delete a particular variable i.e., "ONION" (e.g., a variable n

in the agricultural planning problem described in Sec. 4.4.1) from the model formulation.

In the query (Q : 7.4) the first block obtains constraint type (cons_type) and constraint identifier (cons_id) information from input relation cons_detail (iconsd) for "ROWS" section and stores in file "w2" . Then the second block obtains variable identifier (var_id), constraint identifier (cons_id) and coefficient value (mat_element) from relation matrix (imat) for which variable identifier (var_id) donot match with "ONION*" for "COLUMNS" section and stores the result in file "w4 " . Next the third block obtains the constraint identifier (cons_id) and right-hand-side (rhs value) information from relation cons_detail (iconsd) and stores the result in file "w6 " . Then finally all these external files along with some formatting files are concatenated and processed by interface programs to generate anMPS file for model formulation without the variable "ONION". Similarly other simple analysis as mentioned for other situations can also be made.

(e) Revised Solution with Some Perturbation:

A modeller/analyst may like to analyze the solution after some perturbation done by him in some group of variable values. For example, with reference to sample problem described in Sec. 4.4.1 he may like to decrease the value, say by 3 percent for "FIELD" crops (i.e., "COTTON" or "ONION")

and increase the value, say by 2 percent for rest of the variables and then check for the feasibility associated with present value of variables. Sample query (Q:7.5) illustrates above aspect in Figure 16.

In the query (Q:7.5) the first block initializes the relation "ofunct" by deleting all its previous records. Then next block obtains field values from relation var_value and stores the result in file "f2" which is then used to insert information in relation "ofunct" using next block of query. Then a query block updates the field "ofunct.of4" content by 0.97 times for those records having key match with "COTTON*" or "ONION*". Similarly another query block updates "ofunct.of4" content by 1.02 times for those records having no key match with "COTTON*" and "ONION*". The next query block simply displays the updated information. Finally the last query block displays constraint identifier, computed sum of product of coefficient values and associated updated solution values, constraint type and right-hand-side value for each constraint identifier as described in feasibility check (Q:7.1). In the above manner, the "what-if" analysis of this kind can be made.

(f) Perturbation Study of a Function of the Problem:

A modeller/analyst sometimes may be interested in sensitivity of an objective function which is

derived from the problem structure say a function of some right-hand-side values. For example, he may be interested in the ratio of overall profit from a group of variables to overall profit from another group of variables as a function of some data. With reference to example problem described in section 4.4.1, he may like to increase the objective function coefficient values for variables, say by 2 percent and then like to obtain the ratio of profit contribution from "FIELD" crop (i.e., "COTTON*" or "ONION*") to overall profit and ratio of profit contribution from rest of the variables to overall profit. The sample query (Q:7.6) illustrates the above aspect in Figure 16.

The query (Q:7.6) accomplishes above task using several blocks of queries in sequence. The first block initializes the relation "ofunct" by deleting all its previous records. Then second block of query inserts field values for "ofunct" relation by obtaining the required field information from relation `cost_range (objr)`. The third query block updates field "ofunct.of4" by selecting information from relation `var_value` corresponding to matching variable identifiers. Then the fourth query block updates field "ofunct.of5" by setting the value equal to 1.02 times of the product of coefficient value and associated solution value. Next the fifth block of query inserts a record with "result*" as its key with all other field value initialized.

Then the sixth query block initializes the field "ofunct.of5" by obtaining the sum of the field (ofunct.of4) values for which the key donot match with "result*". Next the seventh query block updates the field (ofunct.of1) by obtaining the sum of profit contribution for variable identifier that match with "COTTON*" or "ONION*", corresponding to record with "result*" as key. The eighth query block similarly updates the field (ofunct.of2) by obtaining the sum of profit contribution for variable identifiers that donot match with "COTTON*" or "ONION*", corresponding to record with "result*" as key. Finally, the last query block displays the profit contribution from field crops, profit contribution from rest of the variables, overall profit, ratio of field crop contribution to overall profit, ratio of rest of variable contribution to overall profit.

Thus the above query (Q:7.6) in Figure 16 provide the information of relative ratio of profit contribution to overall profit contribution.

(g) Tightening or Relaxing an Inequality:

The modeller sometimes may like to tighten or relax one or more inequality constraint(s) due to restrictions or changes in availability of some specific resource. He may like to have analysis of the model solution in light of recent or likely changes. Sample query (Q:7.7) illustrates this aspect in Figure 16.

In the query (Q:7.7) the first query displays all fields of relation `cons_detail` (`iconsn`). The next query block updates all "`cons_type`" of relation `cons_detail` (`iconsd`) to equality ("`E`") i.e., tightens the constraint inequality corresponding to specific constraint identifiers (e.g., "`5*`", "`6*`", "`7*`"). The last query again displays the fields of relation `cons_detail` to illustrate the modification done.

Thus all the sample queries illustrated in Figure 16 provide a wide range of "what-if" analysis a modeller can have for a linear programming model.

Further, all the queries together demonstrate the various data management capabilities of LAMP using the relational format as proposed in our thesis.

The queries cited above are purely indicative of the capabilities of SQL like data manipulation languages and by no means exhaustive. The richness of the queries that can be constructed are only limited to the modeller's innovative capabilities. The queries are not limited to SQL language alone. Many of the relational database management system do support an SQL like language with minor syntactic differences. The ideas illustrated do apply to a more general situation as well. This way we are able to take advantage of the well developed theory of relational databases without we having to write a separate data manipulation language.

CHAPTER VII

SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

7.1 SUMMARY OF WORK:

Our modelling system LAMP accomplishes to a large extent, the objectives we had set out for ourselves in Chapter III. LAMP provides an environment in which an optimization modeller can describe the model structure, obtain the solution of the model, view the results, write reports, perform sensitivity analysis, change parameters and re-solve the problem. Almost all these tasks can be accomplished in an interactive manner using a medium which is natural to the environment. The implementation is achieved using a model specification language to represent the structure and a model manipulation language to manipulate the structure and the data. The model specification language is specially designed as part of this thesis, while the model manipulation is accomplished using a standard query language. To enable the use of a query language, an interface between the problem processor and the model language is also developed in this thesis. Such an interface also introduces an attractive independence between problem processor and modelling languages. In short an integrated

system to support an optimization modeller in all the phases of the modelling life cycle has been developed.

7.2 CONTRIBUTIONS:

The major contributions of this thesis, in our opinion are as follows:

- (a) Demonstrating the feasibility of an integrated system to support optimization modelling
- (b) Development of a modelling language which is simple to learn, powerful enough to model a variety of situations and yet keep close resemblance to the natural way in which modellers perceive the model.
- (c) Outline a possible interface structure between problem processors and modelling languages permitting one to exploit the combined advantages of independent developments in these areas.
- (d) Exploit the powerful capabilities of relational algebraic query languages to permit a variety of "what-if" analysis to be performed.

7.3 LIMITATIONS:

Since our primary aim was to prove the ideas, our implementation is far from perfect. We have not included efficiency considerations in the implementation, which we plan to look into in the near future. Also we have limited

ourselves to optimization modelling, even though we would very much like to enhance the scope to several other related techniques as well. Since, even the extension to general nonlinear programming situation is not very simple, extension to capture other modelling situations does appear to be reasonably hard.

7.4 RECOMMENDATIONS FOR FURTHER WORK:

Any work of this nature in this area is bound to open up several new ideas that can be taken up for further work. Specifically we feel that the following ones are the most promising ones, with a likelihood of significantly changing the modelling scenario:

- (a) Extension to general modelling situations as has been recently proposed by Geoffrion [76]
- (b) Providing explanation capability to the model systems in the line of recent work by Greenberg [83]
- (c) Use of pattern recognition techniques to create a family of Intelligent Modelling Systems.
- (d) Use of Program Generators to assist the task of the design of modelling languages.

REFERENCES

1. Aho, A.V. and Johnson, S.C. (1974), "LR Parsing", Computing Surveys, Vol. 5, No. 2, pp. 99-124.
2. Aho, A.V., Johnson, S.C. and Ullman, J.O. (1975), "Deterministic Parsing of Ambiguous Grammars", Communications of the ACM, Vol. 18, No. 8, pp. 441-452.
3. Alavi, M. and Henderson, J.C. (1981), "An Evolutionary Strategy for Implementing a Decision Support System", Management Science, Vol. 27, No. 11, pp. 1309-1323.
4. Alter, S. (1977), "A Taxonomy of Decision Support System", Sloans Management Review, Vol. 54, pp. 39-56.
5. Alter, S. (1980), "Decision Support System : Current Practice and Continuing Challenges", Addison-Wesley Reading, Ma, USA.
6. Ariav, G. and Ginzberg, M.J. (1985), "DSS Design : A Systemic View of Decision Support", Communications of the ACM, Vol. 28 No. 10, pp. 1045-1052.
7. Armstrong, W.W., Delobel, C. (1980), "Decompositions and Functional Dependencies in Relations", ACM Transactions on Database Systems, Vol. 5, No. 4, pp. 404-480.
8. Astrahan, M.M. and Chamberlin, D.D. (1975), "Implementation of a Structured English Query Language", Communications of the ACM, Vol. 18, No. 10, pp. 580-588.
9. Astrahan, M.M., et.al. (1976), "System R : Relational Approach to Database Management", ACM Transactions on Database System, Vol. 1, No. 2, pp. 97-137.
10. Baker, T.E. (1986), "A Hierarchical/Relational Approach to Modelling", Proceedings of ORSA/TIMS Joint National Meeting, (April, 1986), Session - MC02.2.
11. Beck, L.L. (1980), "A Security Mechanism for Statistical Databases", ACM Transactions on Database Systems, Vol. 5, No. 3, pp. 316-338.
12. Beeri, C. and Bernstein, P.A. (1979), "Computational Problems Related to the Design of Normal Forms for Relational Schemas", ACM Transactions on Database Systems, Vol. 4, No. 1, pp. 30-59.

13. Bennet, J.C. (1983), "Building Decision Support Systems", Addison-Wesley.
14. Bernstein, P.A. (1976), "Synthesizing Third Normal Form Relations from Functional Dependencies," ACM Transactions on Database Systems, Vol. 1, No. 4, pp. 277-298.
15. Bisschop, J. and Meeraus, A. (1977), "Towards a General Algebraic Modelling System", Draft Paper, (July - 77), Development Research Center, World Bank.
16. Bisschop, J. and Meeraus, A. (1979), "Selected Aspects of a General Algebraic Modelling Language", Proceedings of the 9th IFIP Conference on Optimisation Techniques, Warsaw, (Sept. 4-8), pp. 223-233.
17. Blanning, R.W. (1979), "The Functions of Decision Support System", Information Management, Vol. 2, No.3, pp. 87-94.
18. Blanning, R.W. (1983), "What is Happening in DSS", Interfaces, (October), pp. 71-80.
19. Blanning, R.W. (1986), "Tutorial on Model Management", Paper Presented in HICCS 19 - Hawai, January, USA.
20. Blasgen, M.W. and Eswaran, K.P. (1977), "Storage and Access in Relational Databases", IBM Systems Journal, Vol. 5, No. 4, pp. 363-377.
21. Blasgen, M.W., et.al. (1981), "System R : An Architectural Overview", IBM System Journal, Vol. 20, No. 1, pp. 41-62.
22. Bodily, S.E. (1985), "Modern Decision Making - A Guide to Modeling with Decision Support System.", McGraw Hill Book Company.
23. Bodily, S.E. (1986), "Spreadsheet Modeling as a Stepping Stone," Working Paper, Darden Graduate Business School, University of Virginia.
24. Bonczek, R.H., Holsapple, C.W. and Whinston, A.B. (1978), "Aiding Decision Makers with a Generalized DBMS: An Application to Inventory Management," Decision Sciences, Vol. 9, No. 2, pp. 228-245.
25. Bonczek, R.H., Holsapple, C.W. and Whinston, A.B. (1979), "Computer Based Support of Organizational Decision Making," Decision Sciences, Vol. 10, No. 2, pp. 268-291.

26. Bonczek, R.H., Holsapple, C.W. and Whinston, A.B. (1980), "Evolving Roles of Models in Decision Support System," Decision Sciences, Vol. 11, No. 4, pp. 337-356.
27. Bonczek, R.H., Holsapple, C.W. and Whinston, A.B. (1981), "Foundations of Decision Support Systems", Academic Press, New York.
28. Bourne, S.R. (1983), "The UNIX System", Addison-Wesley, Reading, Ma, USA.
29. Boyce, R.F. and Chamberlin, D.D. (1973), "Using a Structured English Query Language as a Data Definition Facility", Technical Report RJ1318, IBM Research Lab., San Jose, CA, USA.
30. Boyce, R.F., Chamberlin, D.O., King III, W.F. and Hammer, M.M. (1975), "Specifying Queries as Relational Expression: SQUARE", Communications of the ACM, Vol. 18, No. 11, pp.
31. Bradley, S.P., Hax, A.C. and Magnanti, T.L. (1977), "Applied Mathematical Programming", Addison-Wesley Pub. Co.
32. Brook, A., Drud, A. and Meeraus, A. (1984), "High Level Modeling Systems and Nonlinear Programming", Discussion Paper, Development Research Center, World Bank, Report No. DRD113, pp. 1-31.
33. Burroughs Corporation (1980), "Model Development Language and Report Writer (MODELER) User's Manual", No. 1094950, Detroit, Michigan, USA.
34. Chadwick, H.N. and Windsor, J.C. (1985), "Decision Support System : A Perspective for Industrial Engineers", IIE Transactions, Vol. 17, No. 1, pp. 38-46.
35. Chamberlin, D.D. and Boyce, R.F. (1974), "SEQUEL: A Structured English Query Language", Proceedings ACM SIGMOD Workshop on Data Description, Access Control.
36. Chamberlin, D.D. (1975), "Relational Database Management Systems", ACM Computing Surveys, Vol. 8, No. 1, pp. 43-66.
37. Chamberlin, D.D. (1980), "A Summary of User Experience with SQL Data Sub-language", Proceedings of International Conference on Data Bases, Aberdeen, Scotland, (July, 1980).

38. Chamberlin, D.D., Astrahan, M.M., Eswaran, K.P., Griffiths, P.P., Lorie, R.A., Mehl, J.W., Reisner, P. and Wade, B.W. (1976), "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control", IBM Journal of Research and Development, Nov. pp. 560-583.
39. Childs, D.L. (1968), "Feasibility of a Set-Theoretic Data Structure— A General Structure Based on a Reconstituted Definition of Relation", Proceedings IFIP Congress, 1968, North-Holland, Amsterdam, pp.162-172.
40. CII - Honeywell Bull (1978), "IDS II Reference Manual", CII - Honeywell Bull, Pef, 46A2AQ88, Rev. 1, Louveciennes, France.
41. Cleaveland, J.C. and Uzgalis, R.C. (1977), "Grammar for Programming Languages", American Elsevier, New York, NY, USA.
42. Codasyl, (1969), "A Survey of Generalized Data Base Management System", CODASYL System Committee, Technical Report ACM, New York.
43. Codd, E.F. (1970), "A Relational Model of Data for Large Shared Data Banks", Communications of the ACM, Vol. 13, No. 6, pp. 377-387.
44. Codd, E.F. (1971), "A Database Sub-language Founded on the Relational Calculus", Proceedings ACM SIGPIDET Workshop on Data Description Access and Control, pp.35-58.
45. Codd, E.F. (1979), "Extending the Database Relational Model to Capture More Meaning", ACM Transactions on Database Systems, Vol. 4, pp. 397-434.
46. Cohen, C. and Stein, J.R. (1978), "Multi Purpose Optimization System : User's Guide Version 2", Manual No. 320, Vogelback Computing Center, North Western University, USA.
47. Control Data Corporation (1979), "APEX-III Reference Manual", Version 1.2, No. 76070000, Rev. G, Minneapolis, USA.
48. Crawford, R.G. and Becker, I. (1986), "A Novice User's Interface to Information Retrieval System", Information Processing Management, Vol. 22, No. 4, pp. 299-308.

49. Cunningham, K. and Schrage, L. (1986), "Optimization Models with Spreadsheet Programs", Working Paper, University of Chicago, USA.
50. Date, C.J. (1977), "An Introduction to database Systems", (2nd Ed.), Addison-Wesley Publishing Company.
51. Date, C.J. (1983), "Database: A Primer", Addison-Wesley Publishing Company.
52. Day, R.E. (1982), "Magic, Version 1.0: A Matrix Generator Instruction Converter for Linear and Integer Programming Models", Department of Business Studies, University of Edinburgh, Edinburgh, Scotland.
53. Delobel, C. (1978), "Normalization and Hierarchical Dependencies in the Relational Data Model", ACM Transactions on Database Systems, Vol. 3, No. 3, pp. 201-222.
54. Delobel, C. and Adiba, M. (1985), "Relational Database Systems", North Holland.
55. Denny, G.H. (1977), "An Introduction to SQL, A Structured Query Language", Technical Report RA93(28099), IBM Research Lab., San-Jose, CA, USA.
56. Digital Equipment Corporation, (1976), "DEC System-10 Programmer's Reference Manual", Maynard, Massachusetts, USA.
57. Digital Equipment Corporation, (1976), "DEC System-10 Database Management System Programmer's Procedure Manual", DEC-110, APPMA - B - 10, Maynard, Massachusetts, USA.
58. Dolk, D.R. and Konsynski, B.R. (1984), "Knowledge Representation for Model Management System", IEEE Transactions on Software Engineering, Vol. SE-10, No.6, pp. 619-628.
59. Dolk, D.R. (1986), "A Generalized Model Management System for Mathematical Programs", To appear in ACM Transactions on Mathematical Software.
60. Earley, J. (1970), "An Efficient Context Free Parsing Algorithm", Communications of the ACM, Vol. 13, No.2, pp. 94-102.

61. Ellison, E.F.D. and Mitra, G. (1982), "UIFP: User Interface for Mathematical Programming", ACM Transactions on Mathematical Software, Vol. 8, No. 3, pp. 229-255.
62. Elson, M. and Rake, S.T. (1970), "Code Generation for Large Language Compilers", IBM System's Journal, Vol.9, No. 3, pp. 166-188.
63. Erikson, W.J. and Hallink, O.P., (1985), "Computer Models for Management Science", Addison-Wesley Publishing Co.
64. Fagin, R. (1979), "Normal Forms and Relational Database Operations", Proceedings ACM SIGMOD International Conference on Management of Database, (May, 1979).
65. Forrest, J.J.H. (1986), "A Minimalist Approach to a Modeling Language", Presented at TIMS/ORSA Joint National Meeting, April, 1986, Session - MC05.1.
66. Fourer, R. (1978), "XML Modeling Language for Linear Programming: Specification and Examples", Working Paper 1006-78, Sloan School of Management, M.I.T., Cambridge, Massachusetts, USA.
67. Fourer, R. and Harrison, M.J. (1978), "A Modern Approach to Computer Systems for Linear Programming", M.I.T. Sloan School of Management, Working Paper No. 988-78, pp. 1-58.
68. Fourer, R. (1983), "Modelling Languages Versus Matrix Generators for Linear Programming", ACM Transactions on Mathematical Software, Vol. 9, No. 2, June, pp.143-183.
69. Friedberg, L.M. (1967), "RPG : The Coming of Age", Datamation, June, pp. 29-31.
70. Gass, S.I. (1982), "Documentation for a Model", ACM Communications, Vol. 25, No. 2, pp. 728-733.
71. Gass, S.I. (1983), "What is a Computer Based Mathematical Model", Mathematical Modelling, Vol. 4, pp. 467-472.
72. Gass, S.I. (1985), "Managing the Modelling Process", Working Paper MS/S 85-002, College of Business and Management, University of Maryland at College Park, Maryland, USA.

73. Gass, S.I., Greenberg, H.J., Hoffman, K.L. and Langley, R.W. (1986), "Impacts of Micro-computer on Operations Research, Interfaces, North-Holland.
74. Geoffrion, A.M. (1983), "Can MS/OR Evolve Fast Enough" , Interfaces, Vol. 13, pp. 10-25.
75. Geoffrion, A.M. (1985), "Structured Modeling : A Progress Report", 12th International Symposium on Mathematical Programming, (August, 1985).
76. Geoffrion, A.M. (1986), "An Introduction to Structured Modeling", Working Paper No. 338, Western Management Science Institute, Univ. of California.
77. Gill, P.E., Murray, W., Saunders, M.A. and Wright, M.H. (1983), "User Guide for SOL/NPSOL : A Fortran Package for Nonlinear Programming", Rep. SOL-83-1, Stanford University, CA, USA.
78. Golden, B.L. and Wasil, E.A. (1986), "Nonlinear Programming on Microcomputer", To appear in Computers and Operations Research.
79. Goldfarb, D. and Mehrotra, S. (1986), "A Relaxed Version of Karmarkar's Method", Presented at ORSA/TIMS Joint National Meeting, October, 1986, Session - MBO3.1.
80. Gray, P. (1983), "Student Guide to IFPS (Interactive Financial Planning System)", McGraw-Hill Book Company.
81. Greenberg, H.J. (1979), "A Tutorial on Computer Assisted Analysis", Frontiers in Operations Research, pp. 213-249.
82. Greenberg, H.J. (1983), "A Functional Description of ANALYZE : A Computer - Assisted Analysis System for Linear Programming Models", ACM Transactions on Mathematical Software, Vol. 9, No. 1, March, pp. 18-56.
83. Greenberg, H. (1985), "Intelligent Mathematical Programming System", Paper Presented in ORSA/TIMS, October.
84. Grotschel, M., Lovasz, L. and Schrijver, A. (1981), "The Ellipsoid Method and Its Consequences in Combinatorial Optimization", Combinatorica, Vol. 1, No. 2, pp.169-197.
85. Harrison, W. (1977), "A New Strategy for Code Generation - the General Purpose Optimization Compiler", Proceedings of 4th ACM Symposium on Principle of Programming Languages, pp. 29-37.

86. Haverly Systems, Inc. (1977), "MaGen: Reference Manual", Denville, N.J., USA.
87. Held, G.D., Stonebraker, M.R. and Wong, E. (1975), "INGRESS - A Relational Database System", Proceedings AFIPS, NCC 44, pp. 409-416.
88. Honeywell Information Systems (1976), "I-D-S/II Programmer's Reference Manual", DE09, Waltham, Mass, USA.
89. Honeywell Information Systems (1979), "Mathematical Programming Systems (MPS) User's Guide", No. DG10A, Rev.0, Waltham, Mass, USA.
90. IBM Corporation, (1971), "MPS/360 Version 2, Linear Separable Programming User's Manual", No.GH20-0476.
91. IBM World Trade Corporation, (1972), "Matrix Generator and Report Writer (MGRW) Program Reference Manual", No. SH19-5014, New York.
92. IBM World Trade Corporation, (1976), "IBM Mathematical Programming Systems Extended (MPS/370) Program Reference Manual", No. SH19-1095-1.
93. IBM (1978a), "Query by Example Users Guide", SH20-2078-0, IBM, White Plains, New York.
94. IBM (1978b), "IMS/VS Publications", GH20-1260-SH20-9025, SH20-9026 and SH20-9027, IBM, White Plains, New York.
95. IBM (1981), "SQL/Data System, General Information", GH24-5012-0, SQL/Data System, Concepts and Facilities , GH-5013-0, IBM Data Processing Division, White Plains, New York.
96. Jarke, M. and Koch, J. (1984), "Query Optimisation in Database Management System", ACM Computing Survey, Vol.16, No. 2, pp. 111-152.
97. Jarke, M. and Vassiliou, Y. (1985), "A Framework for Choosing a Database Query Language", ACM Computing Surveys, Vol. 17, No. 3, pp. 313-340.
98. Jensen, K. and Wirth, N. (1985), "Pascal Users Manual and Report", 3rd ed., Springer-Verlag, New York.

99. Jernigan, R., Hamill, B.W. and Weintraub, D.M. (1985), "The Role of Language in Problem Solving - 1", North Holland.
100. Johnson, S.C. (1975), "Yacc: Yet Another Compiler Compiler", Computing Science Technical Report No. 32, 1975, Bell Laboratories, Murray Hill, NJ 07974, USA.
101. Karmarkar, N. (1985), "A New Polynomial-Time Algorithm for Linear Programming", NAD 014, ATNT Bell Labs., USA.
102. Kashyap, R.L. and Abhyankar, R.B. (1979), "Semantics for Data Retrieval in Relational Database Systems", Proceedings, COMPSAC 79, pp. 319-324.
103. Katz, S., Risman, L.J. and Rodeh, M. (1980), "A System for Constructing Linear Programming Model", IBM Systems Journal, Vol. 19, No. 4, pp. 505-520.
104. Keen, P. and Scott Morton, M. (1978), "Decision Support System: An Organizational Perspective", Addison-Wesley.
105. Kendrick, D.A. and Meeraus, A. (1985), "GAMS: An Introduction", Draft Book, The World Bank, February.
106. Kent, W. (1983), "A Simple Guide to Five Normal Forms in Relational Database Theory", Communications of the ACM, Vol. 26, No. 2, pp.80-84.
107. Kernighan, B.W. and Ritchie, D.M. (1978), "The C Programming Language", Prentice-Hall, NJ, USA.
108. Ketron, Inc. (1975), "MPS III DATAFORM: User Manual", Arlington, Va, USA.
109. Kim Won, (1979), "Relational Database Systems", Computing Surveys ACM, Vol. 11, No. 3, pp. 185-211.
110. Kurator, M. and O'Neill, R.P. (1980), "PERUSE: An Interactive System for Mathematical Programs", ACM Transactions on Mathematical Software, Vol. 6, No. 4, pp. 489-509.
111. Lasdon, L.S. and Waren, A.D. (1979), "Generalized Reduced Gradient Software for Linearly and Nonlinearly Constrained Problem", in Design and Implementation of Optimization Software, H.J. Greenberg (Editor), Sijthoff and Noordhoff, pp. 363-397.

112. Lesk, M.E. (1975), "Lex - A Lexical Analyzer Generator", Computer Science Technical Report No. 39, Bell Laboratories, NJ, USA.
113. Liang, T-P. (1985), "Integrating Model Management with Data Management in Decision Support System", Decision Support System, Vol. 1, pp. 221-232.
114. Liebman, J., Lasdon, L., Schrage, L. and Waren, A. (1986), "Modelling and Optimization with GINO", The Scientific Press, Palo Alto, California.
115. Lozinskii, E.L. (1980), "Construction of Relations in Relational Databases", ACM Transactions on Database Systems, Vol. 5, No. 2, pp. 208-224.
116. Marsten, R.E. (1981), "The Design of XMP Linear Programming Library", ACM Transactions on Database Systems, Vol. 7, pp. 481-497.
117. Martin, J. (1977), "Computer Database Organisation", (2nd Ed.), Prentice Hall.
118. Mccosh, A.M., Scott Morton, M. and Michael, S. (1978), "Management Decision Support Systems", Macmillan Press.
119. McDonald, N. and Stonebraker, M.R. (1975), "CUPID - The Friendly Query Language", Proceedings ACM Pacific 75, pp. 127-131.
120. McKeeman, W.M., Hurning, J.J. and Wortman, D.B. (1970), "A Compiler Generator Implemented for the IBM System 360", Prentice Hall, NJ, USA.
121. Meeraus, A. (1983), "An Algebraic Approach to Modelling", Journal of Economic Dynamics and Control, Vol. 5, pp. 81-108, (North Holland).
122. Mills, R.E. and Fetter, R.B. (1976), "LMC User's Manual", Yale University, Center for the Study of Health Services, Institution for Social and Policy Studies.
123. Mills, R.E., Fetter, R.B. and Averill, R.F. (1977), "A Computer Language for Mathematical Program Formulation", Decision Sciences, Vol. 8, pp. 427-444.
124. Mills, R.E., Fetter, R.B. and Averill, R.F. (1977a), "Conversational Modelling Language - Reference Manual", Working Paper W7-54, Center for the Study of Health Services, Yale University, New Haven, Connecticut, USA.

125. Minch, R.P. and Burns, J.R. (1983), "Conceptual Design of Decision Support System Utilizing Management Science Models", IEEE Transactions on Syst, Man, Cybern, Vol.SMC-13, No. 4, pp. 549-557.
126. Mulvey, J.J. and Zenios, S.A. (1986), "Linking Large Scale Network Optimisation and General Modeling Systems", Presented at ORSA/TIMS Joint National Meeting, October, 1986, Session - TB05.2.
127. Murphy, F.H. (1985), "An Intelligent System for Formulating Linear Programming", Presented at ORSA/TIMS Joint National Meeting, November, 1985, Session - MC19.3.
128. Murtagh, B.A. and Saunders, M.A. (1983), "MINOS 5.0 User's Guide", Technical Report SOL 83-20, Department of Operations Research, Stanford University, USA.
129. Naylor, T.H. (1982), "Decision Support Systems or Whatever Happened to MIS", Interface, Vol. 12, No. 4, pp. 92-94.
130. Nijssen, G.M. (1977), "Modeling in Database Management Systems", North Holland, Amsterdam.
131. Ozon, T.M. (1986), "Applied Mathematical Programming for Production and Engineering Management", Prentice-Hall.
132. Palmer, K.H. (1984), "A Model Management Framework for Mathematical Programming", John Wiley.
133. Parker, B.J. and Ghassan, P. (1986), "Decision Support System: The Reality that Seems Hard to Accept", OMEGA-IJMS, Vol. 14, No. 2, pp. 135-144.
134. Plane, D.R. (1986), "Quantitative Tools for Decision Support Using IFPS", Addison-Wesley, Reading, MA, USA.
135. Raver, N. and Hubbard, G.U. (1977), "Automating Logical Database Design: Concepts and Applications", IBM Systems Journal, Vol. 16, No. 3, pp. 287-312.
136. Rissanen, J. (1977), "Independent Components of Relations", ACM Transactions on Database Systems, Vol. 2, No. 4, pp. 317-325.
137. Rivett, P. (1980), "Model Building for Decision Analysis", John Wiley.

138. Roy, A. and Lasdon, L.S. (1983), "On Detection of Linear and Nonlinear Optimization Problems," Operations Research Letter 2, pp. 149-154.
139. Roy, A., Defalomer, L. and Lasdon, L.S. (1983), "An Optimization Based Decision Support System for Product Mix Problem", Interface, Vol. 12, pp. 26-33.
140. Roy, A., Lasdon, L.S. and Lordeman, J. (1986), "Extending Planning Languages to Include Optimization Capabilities", Management Science, Vol. 32, No. 3, pp. 360-373.
141. Sandberg, G. (1981), "A Primer on Relational Database Concepts", IBM Systems Journal, Vol. 20, No. 1, pp.23-40.
142. Schitlkowski, K. (1985), "EMP : A Software System Supporting the Numerical Solution of Mathematical Programming Problem", Working Paper, Institut fur Informatik, Universitat Stuttgart.
143. Schmidt, J.W. (1977), "Some High Level Language Constructs for Data of t-type Relation", ACM Transactions on Database Systems, Vol. 2, No. 3, pp. 247-261.
144. Schrage, L. (1981), "Linear Programming Models with LINDO", The Scientific Press, Palo Alto, California.
145. Schrage, L. (1986), "Linear, Integer, and Quadratic Programming with LINDO User's Manual", The Scientific Press, Palo Alto, California.
146. Scicon Computer Services Ltd. (1975), "Scicon Mathematical Programming Software", Milton Keynes, England.
147. Scicon Computer Services Ltd., (1977), "MGG User Guide", Milton Keynes, England.
148. Sharda, R. (1984), "Linear Programming on Microcomputers: A Survey", Interfaces, Vol. 14, No. 6, pp. 27-38.
149. Sharda, R. (1985), "A Summary of OR/MS Software on Microcomputers", Working Paper No. 85-8, College of Business Administration, Oklahoma State University, USA.
150. Sibley, E.H. (1976), "The Development of Data Base Technology", Guest Editors Introduction to ACM Computing Surveys, Vol. 8, No. 1, : Special Issue on Database Management Systems, March, 1976.

151. Simon, H.A. (1977), "The New Science of Management Decisions", Prentice Hall.
152. Smith, H.C. (1985), "Database Design: Composing Fully Normalised Tables from a Rigorous Dependency Diagram", Communications of the ACM, Vol. 28, No. 8, pp.828-838.
153. Sneiderman, B. (1977), "Design, Development and Utilization Perspectives on Database Management Systems", Information Processing Management, Vol.13, No. 2, pp. 23-33.
154. Sperry Univac Computer Systems, (1977), "GAMMA 3.4 Programmer Reference", No. UP-8199, Rev. 1, St.Paul, Minn., USA.
155. Sprague, R.H. (1980), "A Framework for Development of Decision Support System", MIS Quarterly, Vol. 4, No.4, pp. 1-26.
156. Sprague, R.H. and Carlson, E.D. (1982), "Building Effective Decision Support System", Prentice Hall.
157. Sprague, R.H. and Watson, H.J. (1986), "Decision Support Systems - Putting Theory into Practice", Prentice-Hall.
158. Steinberg, D.I., (1977), "ALPS (Advanced Linear Programming System): An Easy to Use Mathematical Programming Package", ORSA/TIMS Joint National Meeting, Atlanta, Ga., 1977.
159. Stemple, D.W., Sheard, T.E. and Bunker, R.E. (1986), "Incorporating Theory into Database System Development", Information Processing Management, Vol. 22, No. 4, pp. 317-330.
160. Stevan, L.A. (1983), "Decision Support System: Current Practice and Continuing Trend", Addison-Wesley.
161. Stonebraker, M.R. (1974), "A Functional View of Data Independence", Proceedings, 1974, ACM SIGMOD Workshop on Data Description, Access Control.
162. Stonebraker, M.R., Wong, Kreps, P. and Hald, G. (1976), "THE Design and Implementation of INGRES", ACM Transactions on Database Systems, Vol. 1, No. 3, pp. 189-222.

163. Stonebraker, M.R. (1980), "Retrospection on a Database System", ACM Transactions on Database Systems, Vol. 5, No. 2, pp. 225-240.
164. Tsichritzis, D.C., Lochovsky, F.H. (1983), "Datamodels", Academic Press.
165. Turn, R. and Ware, W.H. (1976), "Privacy and Security Issues in Information Systems", IEEE Transactions on Computers, (November-76).
166. Ullman, J.D. (1980), "Principles of Database Systems", Computers Science Press.
167. Unicom Consultants Ltd. (1977), "UIMP, User Interface for Mathematical Program : Reference Manual", Richmond, Surrey, England.
168. Unify Corporation, (1984), "UNIFY Relational Data Base Management System: Reference Tutorial Manuals", Portland, Oregon, 97219, USA.
169. Unisoft Systems (1983), "UNIPLUS : A System for UNIX Operating System", Vol. I, II, III, Berkeley, California, USA.
170. United Computing Systems, Inc. (1979), "APEX/S4 ALPS Reference Manual", No. 6A16-679.
171. Verhofstad, J.S.M. (1978), "Recovery Techniques for Database Systems", ACM Computing Surveys, Vol. 10, No. 2, pp. 167-178.
172. Wagner, G.R. (1981), "Decision Support System: The Real Substance", Interface, Vol. 11, No. 2, pp. 77-86.
173. Wasserman, A.I. and Prenner, C.J. (1979), "Towards a Unified View of Database Management Programming Languages and Operating System Tutorial", Information System, Vol. 4, No. 2, pp. 119-126.
174. Wang, M.S. and Courtney, Jr. J.F. (1984), "A Conceptual Architecture for Generalized Decision Support System Software", IEEE Transactions on Systems, Man and Cybernetics, Vol. 14, No. 5, pp. 701-710.
175. Waren, A.A., Hung, M.S. and Lasdon, L.S. (1986), "The Status of Nonlinear Programming Software: An Update", to appear in forthcoming Operations Research.

- 176. Williams, H.P. (1985), "Model Building in Mathematical Programming", 2nd Ed., John Wiley.
- 177. Wirth, N. (1976), "Algorithms + Data Structures = Programs", Prentice Hall.
- 178. World Bank, (1982), "General Algebraic Modeling System (GAMS) : Preliminary User's Guide Version 1.0", Development Research Center, The World Bank, N.W., Washington D.C. 20433, USA.
- 179. Zaniola, C. and Melkanoff, M.A. (1981), "On The Design of Relational Database Schemata", ACM Transactions on Database Systems, Vol. 6, No. 1, pp. 1-47.
- 180. Zloof, M.M. (1977), "Query By Example : A Database Language", IBM Systems Journal, Vol. 16, No. 4, pp. 324-343.

APPENDIX A

FURTHER EXAMPLES

FOUNDRY CHARGING PROBLEMS [131]:

A foundry can use four different kinds of scrap alloys of various composition as raw materials to produce a special casting, which will be called here, "cast-A".

Table A-1 and Table A-2 summarize the chemical compound, the cost of raw materials including the melting cost, and the chemical compositions of cast-A to be produced.

Table A-1: Chemical Compound and Cost of Raw Materials.

Raw Material	Chemical Compound in Percent			Cost/ unit wt.
	Aluminium	silicon	Carbon	
Alloy-1	5.0	3.0	4.0	50.0
Alloy-2	7.0	6.0	5.0	70.0
Alloy-3	2.0	1.0	3.0	60.0
Alloy-4	1.0	2.0	1.0	40.0

Table A-2: Chemical Specification of Cast-A to be produced.

Chemical Compound	Atleast in percent	Atmost in percent
Aluminium	3.0	6.0
Silicon	2.0	4.5
Carbon	3.0	6.0

The foundry has a limited supply of Alloy-2 at a level of 400 units and of Alloy-4 at a level of 200 units. The management of the foundry wishes to determine which alloys in what quantities (units) should be included in a 1000 units charge so that they can minimize the raw material cost for producing "Cast-A". The different raw materials availability and total charge required are summarized in Table A-3 below.

Table A-3. Raw Material Availability and Total Charge in Units.

Alloy-1	-
Alloy-2	400.0
Alloy-3	-
Alloy-4	200.0
<hr/>	
Total charge for "Cast-A"	1000.0

The LAMP formulation for the above "Foundry Charging Problem" is given in Figure A.1 and Figure A.2.

GASOLINE BLENDING PROBLEM[131]:

An oil company produces three types of crude oil components - components 1, 2 and 3 - with the specifications shown in Table A-4 below.

* !! NOTE : Lines starting with '*' are 'Comment' lines.

*=====

* TERMINOLOGY PHASE

* MOLECULE

*=====

RAWMAT=AL1,AL2,AL3,AL4

COMPOUND=AL,SI,C

RAWBOUND=AL2,AL4

*=====

* IDENTIFIER

*=====

* RAWMATERIAL COST : (Table A-1)

COST.RAWMAT\$

* COMPOSITION FACTOR : (Table A-1)

COMPOSI.FACTORAT.COMPOUND\$

* MINIMUM REQUIREMENT : (Table A-2)

ATLEAST.COMPOUND\$

* MAXIMUM REQUIREMENT : (Table A-2)

ATMOST.COMPOUND\$

* TOTALCHARGE OF MELTING : (Table A-3)

TOTMELT\$

* BOUNDS : (Table A-3)

CEILING.RAWBOUND\$

Fig. A.1: "Foundry Problem" Terminology Phase.

```

*  !! NOTE : Lines starting with '*' are 'Comment' lines.
*=====

*      PROBLEM STRUCTURE

*=====

*      OBJECTIVE

*=====

MIN(RAWMAT:COST.RAWMAT*RAWMAT?J

*=====

*      CONSTRAINTS

*=====

*      Minimum Constituent Requirement

SUM(RAWMAT:COMPOSE.RAWMAT.COMPOUND*RAWMAT?J =ATLEAST.COMPOUND

*      Totalmelt Requirement

SUM(RAWMAT:RAWMAT?J = TOTMELT

*      Maximum Constituent Requirement

SUM(RAWMAT:COMPOSE.RAWMAT.COMPOUND*RAWMAT?J =ATMOST.COMPOUND

*      Ceilings on Rawmaterial

RAWBOUND ? <= CEILING.RAWBOUND

*=====

```

Fig. A.2: "Foundry Problem" Model Structure.

Table A-4: Crude Oil Component Specifications.

Component	Performance (Octane No)	Vapour Pressure (lb/inch ²)	Production (Units/day)
1	110.0	10.5	40.0
2	100.0	5.0	60.0
3	90.0	12.0	80.0

The company blends these components to obtain gasoline-A and gasoline-B with following specifications:

1. The minimum acceptable performance number of gasoline-A is 90 and of gasoline-B is 100. The performance number of each kind of gasoline is estimated by the weighted average of the performance numbers of the component blended.
2. The maximum acceptable vapour pressure of gasoline-A is 12 lb/inch² and of gasoline-B is 7 lb/inch².

The vapour pressure of each kind of gasoline is estimated by the weighted average of the vapour pressures of the component blended.

How should the company blend the three components to produce gasoline A and gasoline B such that total daily profit is maximized. Components 1,2 and 3 are produced at the required levels of 40, 60 and 80 units per day respectively so that the corresponding requirement constraints are binding constraints with no slack. At present gasoline A sells at a profit of 5 per unit and B at 8 per unit.

Gasoline features required are summarized in Table A-5 below.

Table A-5: Gasoline Features Requirement.

Type	Unit Profit	Performance Number (minimum)	Vapour Pressure (maximum) lb/inch ²
A	5.0	90.0	12.0
B	8.0	100.0	7.0

The LAMP formulation for the above Gasoline Blending Problem is given in Figure A.3 and Figure A.4.

PRODUCT-MIX PROBLEM IN A BAKERY [131]:

Delight Bread and Bun Co. (DBB) employs two parallel baking lines, one for bread products and one for bun products. Each line is independent of the other, although most of the baking operations within the facility are similar in nature. The study was initiated by the management with the objective of determining the amount of each of three types of bread (1.5 lb. white bread, 1.5 lb. wheat bread, and 1.0 lb. white bread) and two types of buns (hamburger and hot dog buns) to be produced in order to maximize the profit. Both of these products are subject to constraints imposed by the capacities of individual machines in the production lines and the consumer demand for the products. Based on the forecast information

```

*  ' NOTE : Lines starting with '*' are 'Comment' lines.
*=====
*      TERMINOLOGY PHASE
*      MOLECULE
*=====

CRUDETYPE=1,2,3

GASOLINE=A,E

GASA=A1,A2,A3

GASB=B1,B2,B3

CRUD1=A1,B1

CRUD2=A2,B2

CRUD3=A3,B3

*=====
*      IDENTIFIER
*=====

*      UNIT PROFIT :                      ( Table A-5 )
PROFIT.GASA$
PROFIT.GASB$

*      PERFORMANCE FACTOR :              ( Table A-4 )
PERFFACT.GASA$
PERFFACT.GASB$

*      Performance Number Required :      ( Table A-5 )
ORA$
ORI$

*      Vapour Pressure Factor            ( Table A-4 )
VFFACT.GASA$
VFFACT.GASB$

*      Maximum Vapour Pressure           ( Table A-5 )
MAVFGASA$
MAVFGASB$

*      Daily Production                  ( Table A-4 )
DFRODL1$
DFRODL2$
DFRODL3$

*      Performance Factor Balance
BFFGASA$
BFFGASB$
*      Vapour pressure balance
BVFASA$
BVFASB$
*      Unit Factor
UNITFACT.GASA$
UNITFACT.GASB$

```

Fig. A.3: "Gasoline Blending" Terminology Phase.

```

*  !! NOTE : Lines starting with '*' are 'Comment' lines.
*=====
*      PROBLEM STRUCTURE
*=====
*      OBJECTIVE
*=====
MAX[GASA:PROFIT.GASA*GASA?]+[GASE:PROFIT.GASE*GASE?]
*=====
*      CONSTRAINTS
*=====
*      Octane Number Requirement for Gasoline-A
SUM[GASA:PERFFACT.GASA*GASA?]+ALPHA[GASA:UNITFACT.GASA*GASA?] =BFFGASA
*      Octane Number Requirement for Gasoline-E
SUM[GASE:PERFFACT.GASE*GASE?]+ALPHA[GASE:UNITFACT.GASE*GASE?] =BFFGASE
*      Vapour Pressure Requirement for Gasoline-A
SUM[GASA:VFFACT.GASA*GASA?]+ALPHA[GASA:UNITFACT.GASA*GASA?] =BVPFGASA
*      Vapour Pressure Requirement for Gasoline-E
SUM[GASE:VFFACT.GASE*GASE?]+ALPHA[GASE:UNITFACT.GASE*GASE?] =BVPFGASE
*      Daily Production of Crud1
SUM[CRUD1:CRUD1?] = DPROD1
*      Daily Production of Crud2
SUM[CRUD2:CRUD2?] = DPROD2
*      Daily Production of Crud3
SUM[CRUD3:CRUD3?] = DPROD3
*=====

```

Fig. A.4: "Gasoline Blending" Model Structure.

the estimated weekly bread and bun demand are as follows:

White bread (1.5):

This is a stable product whose minimum should be greater than or equal to 280000 loaves per week. Maximum should be limited to 300000 loaves per week.

Wheat Bread (1.5 lb):

Wheat bread is not as popular as white bread and is produced mainly to demonstrate a complete bread product line. A maximum production rate of 120000 loaves per week is considered to be realistic.

White Bread (1.0 lb):

This smaller loaf of white bread is not as popular as the larger loaf (1.5 lb) and hence the maximum demand is expected to be at a level of 40000 loaves per week.

Hot Dog Buns:

The maximum demand constraint is 33000 packages per week, the minimum demand is 15000 packages per week.

Hamburger Buns:

The maximum output constraint is 80000 packs per week, the minimum output is restricted to 20000 packs per week. The yearly profit on sales figure are shown in Table A-6.

Table A-6: Yearly Profit on Sales.

Product	Profit	
	Cent per product	Dollars per 10000 product
1.5 lb. white bread	5	500
1.5 lb. wheat bread	4.5	450
1 lb. white bread	3.5	350
Hot dog buns	5	400
Hamburger Buns	4	400

The first group of operations performed in the production of bread (blending, dough formulation, mixing and fermentation) are best grouped together as one operation because of their inter-relationships and because their capacities are difficult to isolate. Thus, all processes performed by these machines will be referred to as Stage 1 of the production. In like manner, the dividing/rounding, overhead proofer, molder and proofer machines are identified as stage 2. Stage 3 contains the oven, cooler and slicing and bagging machines. The definition of these three stages is also applicable to the two bun processes. A capacity unit in hours required to produce 10000 loaves of bread and 10000 packages of buns was found a convenient capacity measure to be used in formulating the machine capacity constraints. The capacity requirements of major stages of production is shown

in Table A-7 below.

Table A-7: Capacity Requirement in Hours per 10000 Units.

Product	Stage 1	Stage 2	Stage 3
1.5 lb. White Bread	4.3	1.2	1.3
1.5 lb. Wheat Bread	4.3	1.2	2.0
1 lb. White Bread	2.9	1.2	1.3
Hot Dog Buns	9.0	2.6	4.0
Hamburger Buns	11.2	2.6	4.2

The times shown include the downtime allowance required for maintenance, cleaning, change over from one kind of product to the other on each line loading and unloading activities. The capacities available for each line and each stage for a three shift work force is shown in Table A-8 below.

Table A-8: Available Capacity for Each Line in Hours.

Product	Stage 1	Stage 2	Stage 3
Bread	135	153	151
Bun	150	135	150

Using above information the management wishes to determine the best product mix for the company.

The LAMP formulation for the above product mix problem is given in Figure A.5 and Figure A.6.

* !! NOTE : Lines starting with '*' are 'Comment' lines.

*=====

* TERMINOLOGY PHASE

* MOLECULE

*=====

BREAD=X1,X2,X3

BUN=X4,X5

STAGE=1,2,3

*=====

* IDENTIFIER

*=====

* UNIT PROFIT : (Table A-6)

PROFIT.BREAD\$

PROFIT.BUN\$

* CAPACITY REQUIREMENT : (Table A-7)

REQ.BREAD.STAGE\$

REQ.BUN.STAGE\$

* AVAILABLE CAPACITY : (Table A-8)

AVLBREAD.STAGES\$

AVLBUN.STAGES\$

* BOUNDS :

UPBOUND.BREAD\$

LOWBOUND.BREAD\$

UPBOUND.BUN\$

LOWBOUND.BUN\$

Fig. A.5: "Bakery Problem" Terminology Phase.

```

*  !! NOTE : Lines starting with '*' are 'Comment' lines.
*=====
*      PROBLEM STRUCTURE
*=====
*      OBJECTIVE
*=====
MAXIBREAD:PROFIT.BREAD*IBREAD? + [BUN:PROFIT.BUN*BUN?]

*=====
*      CONSTRAINTS
*=====
*      Capacity Utilization for Bread
SUMIBREAD:REQ.BREAD.STAGE*BREAD? = AVLBREAD.STAGE

*      Capacity Utilization for Bun
SUMIBUN:REQ.BUN.STAGE*BUN? = AVLBUN.STAGE

*      Ceiling on Bread Production
IBREAD = UPPERBOUND.BREAD

*      Ceiling on Bun Production
IBUN = UPPERBOUND.BUN

*      Minimum Production of Bread
BREAD? = LOWERBOUND.BREAD

*      Minimum Production of Bun
BUN? = LOWERBOUND.BUN
*=====

```

Fig. A.6: "Bakery Problem" Model Structure.

LAMP "HELP" FILE

This document describes the usage of the LAMP (Language Aided Mathematical Programming) system, based on PASCAL language and is capable of generating Linear Programming models.

The salient features of the system are:

1. It is interactive and user friendly, i.e. the system issues messages with prompt for required information, or syntactical error and execution, continues only after acceptable information is supplied by user in the designed format. So to say that error correction is done then and there. The terminal messages with prompt are adequate for input and explanation required to use the system.
2. The syntax of the system is quite similar to algebraic notation to help the modeller.
3. The three basic phases of the model development are Terminology, Database and Abstract model.
4. It accepts declaration of abstract model in Erlang's like language and develops the model in algebraic form.
5. Identifiers can be described in abstract form using molecules (terms). The system generates all possible combinations in the pattern of cartesian product and issues prompt for each identifier data value entry.

identifier# : for data values

identifier@ : for identifier variable name

identifier? : for describing identifier variable

6. Novice user can go for interactive session or partial data entry upto terminologs and database phase through file and user with some LP background may provide information pertaining to all the phases through MIC file.

7. In the abstract model development phase objective function and whole bunch of constraints are developed using identifiers and molecules. Identifiers not defined earlier can also be declared now.

The various modules of the system perform the various steps of model development as described below :

1. TERMINOLOG :

Enters terms of Terminology phase.

2. CHECK TERMINOLOG :

Checks the term for blanks in terms error.

3. ATOM :

Enters list of atoms in Terminology phase.

4. IN ATOM :

Sets decision variable in atomic form.

5. LATTER ATOM :

Enters atoms in latter stage.

6. IDENTIFIER :

Enters identifier as typed by user on terminal.

7. CHECK IDENTIFIER :

Checks for the validity of identifier.

8. USE IDENTIFIER :

Sets identifier used as decision variable.

9. DATA IDENTIFIER :

Sets identifier data values as supplied by user.

8.IDOCHECK :

Checks identifier components with respect to declared molecules.

9.IDOCONSTRAINT :

Selects appropriate procedure after checking the constraint type for adding further constraints as described in abstract form.

10.IIDENTCHECK :

Checks the constraint equation identifier match with database these declared identifiers.

11.SPIDENT :

Accepts identifier in special cases as suggested by user.

12.IDOPRIOR :

Prior check of syntax for constraint equation type.

13.FEASIBLE :

Feasible RHE expression identifier of constraint equation.

14.CONSET :

Check for setting constraint after validation of identifier and type.

15.IDOBOUND :

Generates constraint of bound type ie. $X(j) = b1$.

16.SUMTYPECONS :

Generates constraints of summation type ie. $\sum X(j) = b1$.

17.GENCONS :

Generates constraints in general expanded form.

18.IDOMATGEN :

Writes generated equation for visual aid to user.

19.GCONSTNAME :

Accepts constraint names for general type constraints.

20.GENTYPECONS :

Generates general type constraints which are periodic in nature ie. $X(j) + X(j-1) = b1$.

21.DOFILE :

Terminology and database phase input through separate external file.

21.s DATAFILE :

Accepts the data values for separate external file.

22.COMPRESS :

Compresses the identifier name removing blanks.

23.EQUATIONFORM :

Generates expanded equation form of model developed by LAMP.

24.GFMPS :

Generates external file (F1) having model formulation in industry standard MPS format.

MODEL DEVELOPMENT Algorithm :

Step-1 : Declare Molecules.

Step-2 : Declare Atoms under each molecules if any.

Step-3 : Declare decision variable if other than molecule.

Step-4 : Declare required identifier and enter the values.

Step-5 : Declare objective function and whole bunch of constraints in abstract form.

SYNTAX for objective function/constraints:

<REL> could be <= or = or >=
objective function

MAX or MIN [dvar : identifier * dvar?]

constraints

molecule ? <REL> identifier

SUM [molecule : molecule ?] <REL> identifier

SUM [molecule : identifier * molecule ?] <REL> identifier

Example session : Product-Mix Problem.

1.Declare Molecule : PRODUCT,COMPONENT

2.Declare Atoms for each, if any :

PRODUCT=PA,PB,PC,PI,PE

COMPONENT=RESISTOR,CAPCITOR,TFORMER,SPEAKER,ISISTOR

3.Declare decision variable, if any :

```

4.Declare identifier :
    REQ.PRODUCT.COMPONENT$
    PROFIT.PRODUCT$
    INVENTORY.COMPONENT$
    CEILING.PRODUCT$

```

```

5.Declare objective function and constraint :

```

```

MAX[PRODUCT:PROFIT.PRODUCT*PRODUCT?]
SUM[PRODUCT:REQ.PRODUCT.COMPONENT*PRODUCT?]<=INVENTORY.COMPONENT
PRODUCT? <= CEILING.PRODUCT

```

```

-----
! INPUT FILE CPP2 CONTENTS ARE GIVEN .....

```

```

PRODUCT=PA,PE,PC,PD,PE
COMPONENT=RESISTOR,CAPCITOR,TFORMER,SPEAKER,TSISTOR
*      identifiers .....
REQ.PRODUCT.COMPONENT$
PROFIT.PRODUCT$
INVENTORY.COMPONENT$
CEILING.PRODUCT$
*      objective function and constraints .....
MAX[PRODUCT:PROFIT.PRODUCT*PRODUCT?]
SUM[PRODUCT:REQ.PRODUCT.COMPONENT*PRODUCT?]<=INVENTORY.COMPONENT
PRODUCT? <= CEILING.PRODUCT
*      identifier data values .....
3.0 2.0 1.0 1.0 2.0
4.0 3.0 1.0 1.0 3.0
0.0 2.0 1.0 0.0 2.0
3.0 2.0 1.0 0.0 4.0
5.0 6.0 2.0 2.0 8.0
2.0 4.0 8.0 6.0 2.0
2000.0 1800.0 750.0 500.0 2250.0
200.0 150.0 50.0 400.0 500.0

```

```

-----
! ACTUAL TERMINAL SESSION NOW FOLLOWS ...

```

```

.EX SRIR.PAS
LINK:  Loading
[LINKXCT SRIR execution]

```

```

*
PLEASE TYPE IN ONE OF THE FOLLOWING:
S :TO GIVE THE DATA BY YOURSELF IN THE INTERACTIVE MODE
F :TO TAKE THE INPUT DATA FROM A FILE
>>F
PLEASE TYPE IN THE NAME OF THE FILE (only 10 chr width is permitted)
>>CPP2
FILE NAME :CPP2

```

```

YOU WANT TO ENTER BASIC MODULES :(Y/N) :Y
PLEASE TYPE IN ANY ONE OF THE FOLLOWING FOR WHICH YOU WANT TO DECLARE:
1  FOR 'TERMS'
2  FOR 'ADDING SOME MORE TERMS'
3  FOR 'ATOMS'
4  FOR 'ADDING SOME MORE ATOMS'
5  FOR 'IDENTIFIERS'
6  FOR 'CONSTRAINTS'
7  FOR 'OBJECTIVE FUNCTION'
8  FOR 'GENERAL TYPE CONSTRAINTS'
>>5

```


THE DECLARATION IS DONE FOR ** IDENTIFIERS **
 THE INPUT IS BEING READ FROM THE INPUT FILE :CPFP2

PRODUCT

PA
 PB
 PC
 PD
 PE

COMPONENT

RESISTOR
 CAPCITOR
 TRANSFORMER
 SPEAKER
 TRANSISTOR

YOU WANT TO GO TO OTHER MODULE OF MODEL (Y/N) : Y
 PLEASE TYPE IN ANY ONE OF THE FOLLOWING FOR WHICH YOU WANT TO DECLARE:

- 1 FOR 'TERMS'
- 2 FOR 'ADDING SOME MORE TERMS'
- 3 FOR 'ATOMS'
- 4 FOR 'ADDING SOME MORE ATOMS'
- 5 FOR 'IDENTIFIERS'
- 6 FOR 'CONSTRAINTS'
- 7 FOR 'OBJECTIVE FUNCTION'
- 8 FOR 'GENERAL TYPE CONSTRAINTS'

>>7

THE DECLARATION IS BEING DONE FOR THE OBJECTIVE FUNCTION
 YOU WANT TO DECLARE SPECIAL CONSTRAINT(Y/N):N
 PLEASE TYPE IN THE OBJECTIVE FUNCTION IN THE FOLLOWING WAY
 MAX/MINTERM2:IDENTIFIER*TERM02??

MAXPRODUCT:PROFIT.PRODUCT1*PRODUCT??

YOU WANT TO GO TO OTHER MODULE OF MODEL (Y/N) : Y
 PLEASE TYPE IN ANY ONE OF THE FOLLOWING FOR WHICH YOU WANT TO DECLARE:

- 1 FOR 'TERMS'
- 2 FOR 'ADDING SOME MORE TERMS'
- 3 FOR 'ATOMS'
- 4 FOR 'ADDING SOME MORE ATOMS'
- 5 FOR 'IDENTIFIERS'
- 6 FOR 'CONSTRAINTS'
- 7 FOR 'OBJECTIVE FUNCTION'
- 8 FOR 'GENERAL TYPE CONSTRAINTS'

>>6

THE DECLARATION IS DONE FOR THE CONSTRAINTS
 YOU WANT TO DECLARE SPECIAL CONSTRAINT(Y/N):N
 PLEASE TYPE IN THE CONSTRAINT IN THE FOLLOWING FORMAT
 SUM(TERM0:IDENTIFIER*TERM02??) (<=) OR (=) OR (>=) TERMRHS
 PUT ?? FOR NON-LINEAR TYPE CONSTRAINTS
 PUT > INPLACE OF ? FOR RHS AS VARIABLE

SUMPRODUCT:REQ.PRODUCT.COMPONENT*PRODUCT??<=INVENTORY.COMPONENT

YOU WANT TO DECLARE MORE CONSTRAINTS (Y/N) :Y
 PLEASE TYPE IN THE CONSTRAINT IN THE FOLLOWING FORMAT
 SUM(TERM0:IDENTIFIER*TERM02??) (<=) OR (=) OR (>=) TERMRHS
 PUT ?? FOR NON-LINEAR TYPE CONSTRAINTS
 PUT > INPLACE OF ? FOR RHS AS VARIABLE

PRODUCT??<=CEILING.PRODUCT

YOU WANT TO DECLARE MORE CONSTRAINTS (Y/N) :N
 YOU WANT TO GO TO OTHER MODULE OF MODEL (Y/N) : N
 YOU WANT TO ADD MORE CONSTRAINT DIRECTLY (Y/N) :N

EXIT

PRODUCT-MIX (INVENTORY) PROBLEM [131]:

A manufacture of electronic equipment (equipment items A, B, C, D, and E) finds out that there are certain components in inventory that will not be used in the recently redesigned products to be produced next year. Thus the management desires these components to be used before the end of the year, in the current products, to the best profit advantage. Otherwise they must be shipped to subsidiary plant. The quantities of the components available in the inventory are shown in Table B-1.

Table B-1: Available Components in Inventory

<u>Resistors</u>	<u>Capacitors</u>	<u>Transformers</u>	<u>Speakers</u>	<u>Transistors</u>
2000	1800	750	500	2250

Each product uses certain quantities of components left in the inventory as shown in Table B-2 .

Table B-2 : Components Utilization for Products.

<u>Product</u>	<u>A</u>	<u>B</u>	<u>C</u>	<u>D</u>	<u>E</u>
Resistors	3	4	-	3	5
Capacitors	2	3	2	2	6
Transformers	1	1	1	1	2
Speakers	1	1	-	-	2
Transistors	2	3	2	4	8

A consultation with sales department has indicated that atmost 200, 150, 50, 400 and 500 units of products A, B, C, D and E respectively can be sold within the remaining months of the year. The maximum expected sales and unit profits for products are summarized in Table B-3.

Table B-3: Maximum Expected Sales and Unit Profits.

Product	A	B	C	D	E
Maximum Sales	200	150	50	400	500
Unit Profit	2	4	8	6	2

In order to maximize the profit, how many of each of the five products should be produced so that as many electronic components as possible can be used from the inventory for production.

APPENDIX C

LAMP SYSTEM DOCUMENTATION

The modelling language system LAMP (Language Aided Mathematical Programming) is based on PASCAL and is designed primarily for linear programming modelling situations. LAMP uses "MOLECULE" and "ATOM" in place of set and set element used in the existing modelling languages. In LAMP, variable may be a "MOLECULE" or an "IDENTIFIER" which may be scalar or arrays indexed over the molecules e.g., "WATER.CROP.MONTH" to represent water required for a crop during a month.

The modelling system LAMP has been built up using modular approach. In order to construct the complete formulation of a linear program, various modules of the LAMP system has been designed with subtask mentioned against them. The complete formulation is done in two phases viz. terminology phase and abstract model phase. In the terminology phase, molecules, atoms, concatenated identifiers and decision variables are defined which are used in the next abstract model development phase where objective and set of constraints are constructed for a particular linear program.

The various modules of LAMP are:

- (a) DOMOLECULE: This module defines the molecules of terminology phase and lists all of them after some preliminary checks, using "CHECK MODULE."
- (b) CHECK MOLECULE: This module checks the molecule name for any errors due to the typing and assists the "DOMOLECULE" module in having such validity checks.
- (c) DOATOM: This module defines the list of atoms or elements corresponding to each of the molecules defined earlier by "DOMOLECULE" module and finally lists all the atoms for user verification.
- (d) PSETVAR: This module defines the identifier type decision variable in atomic form using previously defined molecule and atom names.
- (e) ADDATOM: This module adds more atoms in specific situations, if required i.e., in the case of special periodic type of constraint of abstract model.
- (f) READIDENT: This module reads identifier as typed by user in some situation, if not defined or declared earlier during the terminology phase.
- (g) DOIDENT: This module has two sub-modules viz., "DVARIDENT" and "DATAIDENT". Depending upon the declaration i.e., identifier name ending with "@" for "DVARIDENT" and "\$" for "DATAIDENT",

made by user the particular sub-module is activated to fulfil the desired task after checking the validity of the identifier name with respect to declared molecule and atom names, using the module "DOCHECK".

(i) DVARIDENT: This sub-module defines the particular identifier name as decision variable "DVAR" and sets them in compressed atomic form using module "PSETVAR".

(ii) DATAIDENT: This sub-module defines and sets the identifier data values as supplied by the user, during interactive session of modelling or through external file in batch operation.

(h) DOCHECK: This module checks the validity of identifier components with respect to earlier declared molecule names.

With above mentioned modules the terminology phase of model building is accomplished. Following are the modules that assist the user in construction of the abstract model phase of model building. Hereinafter "TERM" and "MOLECULE" are used interchangeably.

(i) SPLCONST: This module deals with special type of constraint situation where objective function or constraint is having two terms in its abstract form e.g., expression or inequality in one of the following types accordingly.

MAX/MIN [TERMO:IDENTIFIER*TERMO2?] +

[TERMOS:IDENTIFIERS*TERMO2S?]

SUM[TERMO:IDENTIFIER*TERMO2?] +

[TERMOS:IDENTIFIERS*TERMO2S?]

<= / = / >= TERMRHS

MAX/MIN [TERMO:IDENTIFIER*TERMO2?] +

ALPHA [TERMOS:IDENTIFIERS*TERMO2S?]

SUM[TERMO:IDENTIFIER*TERMO2?] +

ALPHA[TERMOS:IDENTIFIERS*TERMO2S?]

<= / = / >= TERMRHS

The module "SPLCONST" utilizes its sub-module "DOSPRIOR" for some prior analysis and structure (format) verification of the given special constraint and then develops associated objective or constraint (s) using modules "CONSSET" and "GENCONS". For the objective or constraint structure having "ALPHA" in the abstract model form the user is required to provide the value for "ALPHA" at the time of execution.

(j) DOCONSTRAINT: This module first checks for various structure of constraint type in abstract form i.e., whether it is special, having two terms as described earlier in module "SPLCONST" or having single term. Then using "DOPRIOR", this module does the prior analysis for determining exact

pattern of constraint type such as bound type, simple summation type or general type, matching with only one term of pattern shown in module "SPLCONST". Based on the result of prior analysis, it invokes either the module "DOBOUND" or "CONSSET" and "GENCONS".

The various structures of possible constraint types should be as

$$\begin{aligned} \text{TERMO ?} > = / = / > = \text{TERMRHS} & \quad \text{for bound type} \\ \text{SUM}[\text{TERMO:TERMO ?}] < = / = / > = \text{TERMRHS} & \quad \text{for simple summation type} \\ \text{SUM}[\text{TERMO:IDENTIFIER*TERMO2?}] < = / = / > = \text{TERMRHS} & \quad \text{for general constraint type with} \\ & \quad \text{single term only.} \end{aligned}$$

(k) IDENTCHECK: This module takes the abstract form identifier and checks for its match with earlier declared identifiers and sets some indicator (signal) accordingly for specific use later.

(l) SPIDENT: This module accepts or defines a new identifier in special cases. Whenever an abstract form identifier is not found to match with already declared identifiers the system asks the user for its validity as new identifier. This also uses the module "IDENTCHECK" for verifying identifier components.

(m) DOPRIOR: This module makes prior check of syntax for abstract constraint in row-wise organization and after eliminating blanks, checks for all special characters which are used as indicators for different constraint type determination. If required in some cases, it can provide nonlinear type constraint analysis and parameter value fixation by user.

(n) READRHS: This module reads right hand side (RHS) identifier of abstract form constraint viz. "TERMRHS" and displays the same on terminal for user verification.

(o) CONSET: This module makes identifier component (part) match check with the decision variable of the given abstract form constraint and uses module "IDENTCHECK" for identifier match with already declared identifier list and then depending upon the requirement, invokes modules "READIDENT" or "SPIDENT" and "READRHS" based on specific response from the user. Hence this module sets the various components of abstract form constraint after the validity checks for associated variable, coefficient identifier, right hand side identifier (TERMRHS) and summation index (TERMO) are made.

(p) DOROUND: This module generates simple bound type constraints from the given abstract form

$$\text{TERMO} \leq = / = / > = \text{TERMRHS}$$

This module uses modules "IDENTCHECK" and "READRHS" for verification of "TERMRHS" of given abstract form constraint

and finally writes the corresponding inequalities in expanded form.

(4) GENTYPECONS: This module generates a constraint similar to generalized bound (upper or lower) type from the given abstract form constraint as

$$\text{SUM}[\text{TERMO}; \text{TERMO?}] < = / = / > = \text{TERMRHS}$$

(r) GENCONS: This module uses the sub-module "GENMAT" which sets the constraints in general expanded form. Using the results of module "CONSSET" which checks the part (component) successively, it further checks whether the abstract form constraint identifier matches with the earlier declared identifiers having data values stored in the system. Later, using an index function the proper coefficient values are retrieved from the array containing the data values for particular identifier. Finally, based on "TERMRHS" identifier name the constraint name(s) is also set and stored in the system. This module uses "DOMATGEN" to write the matrix co-efficients.

(s) DOMATGEN: This module writes the generated matrix coefficients for visual aid to user on terminal.

(t) GCONSTNAME: This module sets the name of the constraint as provided by user for the modelling situation using module "GENTYPECONS" dealing with periodic type constraint.

(iv) `TYPECONS`: This module generates periodic constraints having one period lag component. This invokes module `CONSTRAINTS` for labelling various constraints.

(v) `DATAFILE`. This module provides the facility of generating the linear program model in batch mode for a professional user, if he opts for the same instead of interactive session. In the batch mode modelling he should provide all the relevant information needed for model building in certain order through a separate external file having entries as below.

```
*MOLECULE1=ATOM1,ATOM2,...
```

```
CROP=COTTON,ONION,APPLE,ORANGE
```

```
::::::::::::
```

```
*IDENTIFIER(S)
```

```
WATER.CROP.MONTH$
```

```
::::::::::::
```

```
*ABSTRACT MODEL OBJECTIVE/CONSTRAINTS
```

```
MAX[CROP:PROFIT.CROP*CROP?]
```

```
SUM[CROP:WATER.CROP.MONTH*CROP?] <= WATERAVL.MONTH
```

```
::::::::::::
```

```
*IDENTIFIER DATA VALUES IN SAME ORDER OF IDENTIFIER
```

```
95.0
```

```
::::::::::::
```

Thus using sub-module "DATAFILE" this module accepts input through an external file for molecules, atoms and identifiers. Further this module alongwith command file and the same external file having information for objective and constraints in abstract form completes the modelling of linear program based on information constrained in the above mentioned files.

(v) EQUATIONFORM: This module generates the entire model formulation as linear program in expanded form. The whole bunch of constraints alongwith objective function at the top of it is written on a separate file named "F2" which can be utilized further by any problem processor for obtaining solutions to linear program. The problem processor used in our case is LINDO [145]. Further as LINDO does not accept more than 132 characters for input in a line in the form of objective function as well as constraints, the same has been implemented as a specific requirement, which need not be so in general.

(x) GFMPS: This module generates the model formulation in industry standard MPS format [92] acceptable to most of the existing problem processors e.g., LINDO and others. By invoking this module, the model formulation of linear program is written in MPS format on a separate file named "F1" which can be utilized further by any problem processor for obtaining solutions of the modelled linear program.

Thus the major task of model formulation is achieved by the above mentioned modules. However, some small modules also assist in accomplishing the above task viz.,

PREAD : It reads a character from the terminal.
PWRITE : It displays a character on the terminal.
PLNREAD : It reads a character from a new line.
COMPRESS : It compresses the identifier name after removing the blanks.

Apart from the above, LAMP can also generate model formulation for transportation and assignment problem by invoking modules "TRANSPORT" and "ASSIGNMENT" respectively in standard formulation pattern.

APPENDIX D

LINEAR INTERACTIVE AND DISCRETE OPTIMIZER LINDO

LINDO (Linear Interactive and Discrete Optimizer) is an interactive linear, quadratic and integer programming modelling language system developed by Linus Schrage at Graduate School of Business, University of Chicago, USA. It is a small scale system designed to be useful to a variety of users. The major design consideration has been that if a (potential) user wants to do something simple, then there should not be a large set-up cost to learn the necessary features of modelling language system LINDO. LINDO has been in use since 1981 for solving real industrial linear, quadratic and integer programs of respectable size. Several versions of LINDO have been implemented on main frame, mini and micro computers of different computer manufactures viz. DEC, IBM, Data General, Honeywell, Burroughs, CDC, HP, Univac, etc. on DEC-20 computers it is typically configured to solve problems with upto 800 rows and 4500 variables. This capability is generally increasing with each new version of LINDO. It is biased towards the user who is interested in developing and solving formulation of real life problems from diverse fields of military, industries, agriculture, transportation and service sectors.

LINDO is designed for interactive computing. However, it can be run in batch mode with no modification through external file on a batch system. LINDO is command oriented rather than menu oriented. Thus it does not restrict any user to go through fixed sequence of steps allowing only small options along the way. User can select from wide range of commands in any order as per his specific modelling requirement. It can be run in a conversational style. Introductory help about the system features are provided by "HELP", "CATAGORIES" and "COMMANDS" commands. Using them user can get the list of all commands available, which are grouped into categories according to use e.g., input, output, etc. Then using "HELP" followed by any command name he can get the detailed information of specified command.

LINDO user can divert the solution and model formulation of real life problem to a designated external file. LINDO has file output commands for two purposes:

- . Store a problem formulation;
- . Store the results of computation.

User can provide model input to LINDO through external files which may have input information either in industry standard MPS format or in expanded inequality form. LINDO has specific commands for taking input from files. The MPS format for describing an LP is widely accepted industry format.

The MPS format has from three to five sections. "ROWS", "COLUMNS", "RHS", "BOUNDS", "RANGES". "BOUNDS" and "RANGES" are optional.

The "ROWS" section lists the row names, one per line, with the name preceded by its type. An "L" means ($< =$), an "E" means ($=$), a "G" means ($> =$) and an "N" means not constrained (e.g., the objective function row).

The "COLUMNS" section lists each non-zero element of the matrix preceded by the column name and row name in which it appears.

The "RHS" section lists each non-zero right-hand-side element preceded by the name of the row in which it appears. One must also give a name to the right-hand-side i.e., "RHS".

In MPS file form of input it is allowed to have several non-constrained rows i.e., multiple objectives can be listed by assigning "N" alongwith particular row name having objective function. The system asks user to indicate the particular objective function to solve the model formulation through given MPS file alongwith the information regarding maximization or minimization.

LINDO provides an elegant problem summary (e.g., number of rows, number of variables, number of integer variables, number of non-zero elements, number of non-zero constraints,

matrix density, smallest and largest element in absolute value, number of less than equal to constraints, number of equal to constraints, number of greater than equal to constraints, objective function, number of generalized upper bounds, etc.). This summary is quite useful for model analysis of large problems. Model validation before solution is an essential task for a large size linear program. In the case of large linear program model the variables may be misspelt, the sign of coefficient may be wrong, the coefficient may be in wrong constraint, the coefficient may be associated with wrong variable, the decimal point may be misplaced. Such errors may lead to absolutely erroneous solution which necessarily have to be eliminated before model being run for solution. In this context model analysis capability of LINDO in the form of problem summary is commendable.

LINDO is quite useful for mathematicians as it is capable of providing several analysis of working of Simplex method. LINDO uses Revised Simplex Method. Various commands allow user/analyst to study the mechanics of the Simplex method for solving linear programs. One may optionally include a variable name after the word pivot, e.g., "PIVOT COTTON", in which case the named variable (e.g., "COTTON" in this case) is made to enter the solution. One may optionally

include a row number after the variable, in which case the pivot will be made in the specified row. LINDO has the features for the mathematician who may be interested in displays of the tableau and some other steps of Simplex method. Specific commands in LINDO provide insight into the workings of the revised Simplex method, "INVERT" command reinverts the current basis i.e., it re-solves the set of simultaneous linear equations implied by current basis. The inverse is represented by product of elementary matrices (i.e., identity matrix except for one column). Inversion tries to permute the rows and columns of the basis so that the matrix is as close to the triangular as possible because a triangular system of equations is easily solved by back substitution.

In order to study the effect of various parameters on the optimal solution, LINDO provides specific commands. The "RANGE" command causes a standard range report to be printed. This report specifies the allowable changes for objective function and right-hand-side coefficients which will not cause a change in basis. LINDO allows for sensitivity analysis also.

LINDO has a quadratic programming capability which allows the user to consider problems with a quadratic objective function i.e., objectives which contain the

product of few variables. Users in general be interested in whether the optimum found by the quadratic program (QCP) algorithm is global optimum or simply a local optimum. If the submatrix corresponding to the quadratic part satisfies certain conditions then the solution found will be global-optimum. One condition sufficient to guarantee global optimality is positive definiteness.

LINDO allows variables to be integer. It can be general integer variable or zero/one variables. General integer variables are identified by specific command. LINDO uses branch-and-bound method for solution of integer programs.

LINDO contains a dummy subroutine called "USER". This allows a user to do a variety of things, such as write a special purpose input/output procedure or incorporate LINDO into larger systems.

APPENDIX E

UNIFY RELATIONAL DBMS SOFTWARE

"UNIFY" is a relational database management system software developed and supplied for commercial use by UNIFY Corporation, Portland, Oregon, USA. "UNIFY" is based on "UNIX" operating system. UNIFY is a collection of over twenty different programs, all integrated together to let user create and modify application systems that store and retrieve data. The UNIFY System is menu driven. The primary user interface to UNIFY system is the menu handler (MENUH). The menu handler provides the means to let user select any UNIFY program he wants, as well as controlling security, so that he can deny access to certain programs. This menu handler is linked directly with set of utilities viz., "SQL", "Query By Forms", "Enter ", "Database Utilities", "Host Language Interface". All these utilities are directly linked with "Unitrieve Data Base Kernel" which is linked with "Data Dictionary" and "Data Base Volumes". Report writer is linked to "SQL" and "Query By Forms". UNIFY is provided with a set of built-in menus ready for user to use in developing his application. However, he can also create his own menus.

In order to exploit the DBMS capabilities fully the system menu provides following options: schema maintenance, schema listing, create data base, " SFORM" menu, "ENTER" screen registration, "SQL"- Query/DML language, "SQL" screen registration, listing processor, data base test driver, "MENUH" screen menu, "MENUH" report menu, reconfigure database, write data base backup, read data base back-up, data base maintenance menu.

The UNIFY non procedural query and update language is an implementation of the IBM-standard structured Query Language (SQL). Using SQL, user can interactively add, modify, delete and query information in his data-base. Moreover the user can connect a screen form to an SQL query script and then format the results of the query with the primary UNIFY report writer, "RPT". This interface is usually called SQL By Forms. For performing the simpler queries that are often more than half of the queries performed, UNIFY offers Query By Forms (QBF). QBF lets user fill in search values on a screen form, which is used to find the records that match. User can look at them one at a time on the screen form or he can send the results to one of the UNIFY report writers: "RPT" which is powerful, listing processor "LST" which is easy to use.

Full screen data entry is handled by the utility "ENTER". ENTER performs many kinds of edit checks including

type, length, date and time formats, amount formats, and table look-up. In addition, user can add his own functions to customize it further. ENTER uses the screen forms that can be created and modified using the SFORM utilities.

The utilities necessary to create and maintain a database include: programs to enter and modify the database design (i.e., schema design); create and reconfigure the database, maintain field level security; print statistics, read and write back-up tapes (or floppies); repair corrupted data bases; and load in data from ASCII files. The remaining part of the structure are internal to the system. Although operations of UNIFY requires little or no knowledge of UNIX or programming, having some background of above is quite useful.

The transaction logging and console monitor utilities make the operation of application system easier and more reliable by recording all database updates and displaying menus for user guidance respectively.

In order to use UNIFY DBMS Software for the application, user first creates and maintains the schema and the database itself using data base maintenance utilities. Then using the screen form tool SFORM data could be entered in the database. The general purpose data entry program "ENTER" can be used to drive SFORM Screens.

Once the data has been entered in the data base, the UNIFY SQL and RPT processors can be used to retrieve data or present it in a meaningful format. The UNIFY query language, SQL, is a powerful subset of the IBM standard relational query and data manipulation language, SEQUEL 2. Besides SQL, UNIFY also has a simpler, less powerful query interface designed to do file listings with totals and subtotals. This program is called listing processor (LST).

UNIFY can be used as the SHELL. The main shell commands are:

- DBLOAD : This permits loading of data into database from an ASCII file.
- LST : This allows producing simple sorted file listings with totals and subtotals.
- RPT : This allows producing sophisticated reports using a powerful, non-procedural language.
- SQL : This allows query and update of a database file using powerful, non-procedural language.

The UNIFY's specific features of schema design is described below.

The "UNITRIEVE" data base is set-up and maintained by designing interactive schema for appropriate records and their fields using the relational approach. The schema entry program uses two screen forms, first for general record

information (e.g., Line Number - LN, Command - CMD, Record name - RECORD, Expected numbers, Long name - LONG NAME and Description) and the second for detailed field information for the record created earlier.

The various user assistance prompts of paging area are:

- n - displays next page of data base record type.
- p - displays previous page of data base record type.
- a - permits adding of new data base record type.

The various command area options are:

- f - modify the fields for current record type.
- m - modify, expected, long name and description for the current record.
- d - delete current information (record).
- q - redisplay the paging prompt at the bottom of the screen.

Some of the restrictions on the record and field details are as follows:

A record can be given a name of maximum eight characters having upper and lower case letters, numbers and underscore character (_) beginning with letter only. Long name upto sixteen character length with other features similar as record name can be given which is used by "ENTER", "SQL" utilities.

Description can be used as "Comment" field for description of record.

The field information (e.g., Line number - LN, Command - CMD, field name - FIELD, Key - KEY, Reference - REF, Type - TYPE, Length - LEN, Long name - LONG NAME and Combination Field - COMB) can be set for any record using prompt "f" in the command area of record schema maintenance. "Field name" has the same feature as that of record name. Asterisk (*) is the entry to be made in the "KEY" column to indicate a particular field to be treated as key field.

"REF" column is used to establish an explicit relationship with another record type (i.e., linking with some base field.) The "TYPE" column is used to indicate the different data types associated with the field using first character as follows: "n" for "NUMERIC", "f" for "FLOAT", "S" for "STRING", "d" for "DATE", "t" for "TIME", "a" for "AMOUNT" and "c" for Combination (COMB).

The "LEN" column represents the entry for the length of the various fields for display. If not provided by the user at schema design the system assumes some default values. "LEN" has the form "nnd" for float field (i.e., 179 represents 17 display positions having 9 positions for decimal values). "COMB" field column is the unique feature of UNIFY system that it allows user to specify multiple fields as key of the record type (relation).

The "LONG NAME" column is used to give long field names (upto sixteen characters). The regular field name is valid upto eight characters. The long field names do not have to be unique for the whole data base, but the regular field names must be. Field names must begin with letter and comprise of letters, digits and under-score character (_) having no blanks in between. In case of same long field name in different record types the particular field name is referred by prefixing the record type name to the long field name (i.e., iconsn.cons_id or imat.cons_id).

The "COMBINATION" column is used to specify multiple field key, if any.

These features are used to create and maintain schema in a systematic way. The "RETURN" key on terminal is used for forward reference and "CTRL" key along with "U" key is used for backward reference alongwith specific options selected on the display terminal during the UNIFY system operation.

APPENDIX F

STRUCTURED QUERY LANGUAGE

The Structured Query Language (SQL) was developed at the IBM Research Center as a relational inquiry and data manipulation language based on English Keyword Syntax. Its structure was refined through extensive testing to produce a language easy enough for non specialists to use, yet powerful enough for data processing professionals. SQL is fast emerging as the standard relational query language on variety of computers starting from main frame to micros. The UNIFY implementation of SQL is primarily based on the language description [38]. In order to adapt it to super-micros/minis and the UNIX environment some changes have been made to the syntax. Most of the query and data manipulation features have been implemented without modification and UNIFY's SQL is claimed to be the most powerful relational data manipulation language available on supermicros.

SQL is a powerful, English keyword based language that lets user specify queries of varying degrees of complexity. A query consists of phrases (also called clauses), each of which is preceded by a keyword.

These keywords are:

and	asc	avg	between
by	count	delete	desc
edit	end	fields	from
group	having	help	in
insert	into	is	lines
max	min	not	or
order	records	restart	select
separator	set	start	sum
unique	unlock	update	where

The required phrases (clauses) are:

select - Some data (a list of field names)
 from - Some place (a list of record types)

The optional phrases are:

where - a condition (a true/false statement)
 group by - some data (a list of field names)
 having - a group condition (a true/false statement)
 order by - some data (a list of field names)
 into - a file name.

The relations (record types) and fields to be used in selection and calculation are identified by the relation names and long field names (maximum of sixteen characters) from the UNIFY schema design for the database. Schema design

of UNIFY permits both regular short names (upto eight characters) and long names (upto sixteen characters) for naming of the fields. Long field names donot have to be unique for the whole data base; but the regular field names must be. In case of same long field names in different relations the specific field name qualification is done by prefixing the relation names. Valid field names should begin with letter only and comprise of letters, digits and under score character (_) having no blanks in-between.

SQL allows free form input - line can be spaced across/ many statement clauses can be compressed in a line. This means that extra spaces and carriage returns may be used freely to improve the readability of a query statement, without changing its meaning. A query is ended by the slash character (/), at which time SQL begins the evaluation of what user has typed. If the user entered a valid query, the message in response to slash character is "recognised query". If the query had some syntax error and SQL was unable to understand the user query, an error message is displayed on the terminal that gives user an idea as to what kind of error was made by him in writing the query statement.

UNIFY's SQL provides three kinds of help that makes learning and using SQL easier. User can get help about general syntax, any specific keyword, keyword clauses or the valid field names for the data base.

The simplest kind of SQL query includes both a "select" clause and a "from" clause. The "select" clause lists the fields to be printed, while the "from" clause tells which record type (s) or relation (s) the fields are to come from. As user rarely wants to list the entire contents of a record type (relation), the "where" clause is provided to let user specify selection criteria. The "where" clause lets user compare a field with a constant, an expression, or the results of another "select" clause. The "where" clause can also contain a complex boolean expression composed of selection criteria connected by "and"/ "or" operators. Precedence can be established by use of square brackets.

Arithmetic expressions are provided to let the user calculate values using the standard operators +, -, * and /. Both constants and fields may be used in arithmetic expressions, which are allowed on fields of all types except string and combination. Arithmetic may be used wherever a simple field is allowed in "select" , "where" and "having" clauses.

SQL provides five different built-in aggregate functions to allow calculation of aggregate items in a query result. The functions are "count(*)", "min", "max", "sum" and "avg". Aggregate functions are only valid when used in "select" or "having" clauses. User may not use an aggregate function

directly in a "where" clause, although using nested query he can usually achieve the same effect. As natural, an aggregate function only applies to a group of records with a common characteristic. The "group by" clause is provided to allow computation of aggregate function on group of records that have common characteristics. Thus using a "group by" clause without an aggregate function has no meaning. The effect of a "group by" clause is to sort the selected rows by the indicated fields, and then perform the aggregate functions at each level break. The output results are sorted also.

Nested queries allow user to get answer for a whole new set of questions that cannot be answered using the capabilities of SQL presented so far. Nesting allows user to use the results of one query as input to another. Thus he can use the results of one question in answering another one, for example, checking simple upper bound situation in our relational database for linear programming model as below:

```
Select  iconsn.cons_id, cons_name
from iconsn, imat
where imat.cons_id = iconsn.cons_id and iconsn.cons_id^ =
      Select  imat.cons_id
      from    imat
      where   imat.mat_element^ = 1;
group by  iconsn.cons_id
having sum (imat.mat_element) = 1/
```

Queries can be nested to any level and evaluation takes place from innermost to outermost block in order. Nested queries can be used in both "where" and "having" clauses at the same time.

The "having" clause lets the user select some of the groups formed by a previous "group by" clause and reject others based on the results of another selection using one or more of the aggregate function. This gives user a capability equivalent to using an aggregate function in a "where" clause, which is not allowed otherwise.

The "order by" clause lets the user sort the output of a query. He can specify ascending or descending sorts on each field, or can omit the specification entirely. In such situations, the default sort order is ascending, with "STRING" fields sorted in alphabetic order from A to Z.

SQL may list fields from any number of record types (relations) in a single query. Queries that list fields from several record types are called join queries, because they join the different record types together. The different record types (relations) to be involved in the query are listed in the "from" clause, in any convenient order. SQL then determines what is the most efficient method of performing the selection and qualification. The fundamental concept underlying join queries is that of Cartesian product. Conceptually, a join query first forms the Cartesian product of the record types (relations) and then "filters" the result by the conditions in the "where" clause.

Thus a join query without a "where" clause does in fact list the Cartesian product of the record type. Sometimes it is necessary to join a record type (relation) with itself.

Regular nested queries work "from the inside out", evaluating the innermost query and passing the resulting value or values back out to the next outer query. Each query is performed only once. Variable queries reverse this procedure, working "from the outside in". This lets the user perform an inner query multiple times using values from the outer query. As the inner query varies depending on the record type (relation) in the outer query, this kind of query is called as variable query. The inner query to be performed multiple times is ended with a semicolon (;) instead of ending the query with a slash character (/).

In addition to above stated query facilities, SQL also provides data manipulation capabilities. This means user can interactively insert, modify and delete records in his database using a high-level, nonprocedural language. There is also an interface to the UNIFY database load program "DBLOAD", so user can load a database quickly from regular ASCII files. All of the query features of the language are retained. Hence the user can use regular query statements to insert data from existing files containing record types into other files, and select records to be modified or deleted. When a set of records is specified to

be modified, that set is locked using the UNIX file locking system calls. This prevents several users from simultaneously trying to modify (update) the same records.

The "insert" clause allows the user to add new records to the database, using constant value or values returned as the result of a query. Only the primary key value must be specified. Other field values that are not specified are set to the standard, initial values -- zeros for "NUMERIC", "FLOAT", "AMOUNT", "TIME" and "STRING" fields and null (-32768) for "DATE" fields. Besides using literal tuples, an "insert" clause can use the results of a query statement to obtain its values.

The "update" clause lets the user modify fields in existing records. Updates can be specified with literal values, expressions or query statements. A "where" clause to specify which records the update applies to is not required. If it is left out, the update is applied to all records of the indicated type. A field can be updated using an expression, and more than one field can be changed in a single update clause.

The "delete" clause lets the user delete records from an existing file (record type). The records to be deleted are identified by a "where" clause or if this is not present, all the records for the indicated type are deleted.

In addition to above mentioned facilities, UNIFY's SQL provides several extensions to the basic language for interactive query development in the UNIX environment, which makes it more powerful.